NEURAL NETWORKS

FOR CONTROL

By

GISELE GUIMARÃES

Bachelor of Science
Universidade Federal de Goiás
Goiânia, Brazil
1984

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1988

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
July, 1992

NEURAL NETWORKS

FOR CONTROL

Thesis Approved:

Martin T Hagan

Thesis Adviser

Donald G. Whiten

John Wolfe

Jess E Boston

Thomas C. Collins

Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my gratitude to Dr. Martin T. Hagan, my major adviser for his interest, guidance, dedication, counseling, patience, invaluable suggestions and, notably, his friedship. My appreciation is also extended to Dr. Ronald Rhoten, Dr. Keith Teague, Dr. James Baker and Dr. John Wolfe for being members of my committee. Special thanks to Nidal Sammur and Mohammad Menhaj for their help and encouragement, and to all those who made this research possible.

I am most greatful to my husband Jose Vicente for bringing me here and, above all, for his patience and loving support. My thanks to mom and dad for making it possible for us to come, for their many letters and inspirations. Thank you Gilson and Gislene for being helpful and a company to our parents.

I would also like to thank my mother-in-law, my brothers and sisters-in-law for understanding my husband's absence during our studies.

Finally, I would like to thank all persons who somehow contributed to this accomplishment. I hope that my effort in the completion of this work corresponds to theirs.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER I

INTRODUCTION

Design methods of control systems for linear systems are well-known. However, design methods for nonlinear systems are still an active area of research. This research studies the feasibility of using neural networks as controllers for nonlinear systems. A particular nonlinear system, the Extended Range Gun (ERG), has been selected. The adaptive stabilization of the ERG is considered in this research. Two design techniques, using neural network controllers, developed by B. Widrow and K. Narendra have been studied for the stabilization of the ERG.

This report contains 7 chapters. Chapter II explains how neural networks can be used as function approximators and describes in detail the backpropagation algorithm. This algorithm is used in both design techniques mentioned above. Preliminary results showed that training time for this algorithm, even for simple problems, can be excessive. Modifications to the backpropagation algorithm in order to reduce learning time are discussed in Chapter III.

Chapter IV describes how neural networks can be used as system controllers. First, it is assumed that prior information about the input/output characteristics of the controller is known. In this case, function approximation is applied. Next, it is assumed that the

input/output characteristics of the controller are not known. To handle that situation, the techniques developed by Widrow and Narendra are described.

The control design techniques presented in Chapter IV are applied to the ERG. Results of these techniques are shown in Chapters V and VI. They test the feasibility of using neural networks in this application and help determine the best implementation of the algorithms. Chapter V presents the technique developed by Widrow, while Chapter VI contains Narendra's method.

Finally, Chapter VII provides a summary of the findings of this research and gives recommendations for future work.

CHAPTER II

BACKPROPAGATION

Neural Networks

Artificial neural networks are simplified models of biological neural networks found in the brain [1]. They are composed of thousands of neurons connected together. An idealized biological neuron is shown in Figure 2.1. The information arriving from thousands of other neurons are the inputs. These inputs are weighted by the synapses, which connect axons to dendrites. The weighted inputs are summed by the dendrites. The cell body converts this sum into an output through a nonlinear relationship. The output is carried by the axon to other neurons connected to it.

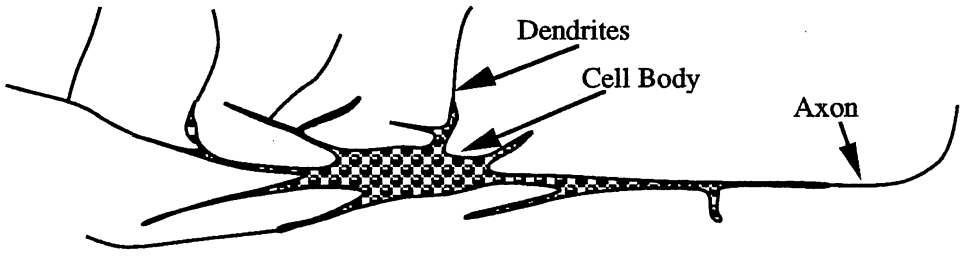


Figure 2.1. Simple Model of a Biological Neuron

The artificial neuron, a single layer perceptron, is shown in Figure 2.2. The x_i's are the inputs. The w_i's are the weights, which mimic the strengths of the connections (synapses). The dendrites and cell body are represented by the sum and the nonlinearity. The nonlinearity modeled throughout this work is shown in Figure 2.2. The output is y (axon). There are other forms of nonlinearities that can be used, for example

$$\frac{1-e^{-x}}{1+e^{-x}} \qquad \text{and} \qquad \frac{1}{\pi}\arctan(x) \tag{II.1}$$

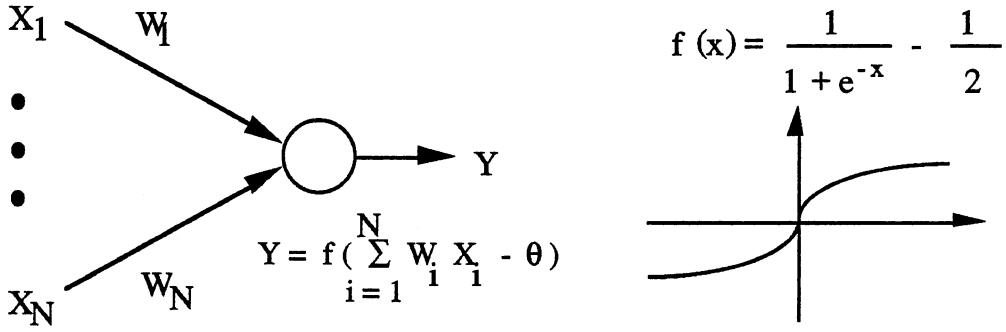


Figure 2.2. Single-Layer Perceptron and Transfer Function

Function Approximation with Neural Networks

Many single layer perceptrons (single "neurons") organized in layers constitute a multilayer perceptron. Figure 2.3 shows a two layer perceptron. A two layer perceptron is capable of

approximating any square integrable function [2,3]. Inputs in a compact set $U \subset \Re^n$ map to outputs in a set $V \subset \Re^m$. The function ghat(w): $\Re^n$ -> $\Re^m$ will approximate g: U -> V by identifying the parameter $w^* \in \Re^k$ (weight vector) which gives the best approximation of g. An example bellow illustrates the power of the multilayer perceptron in implementing functions.

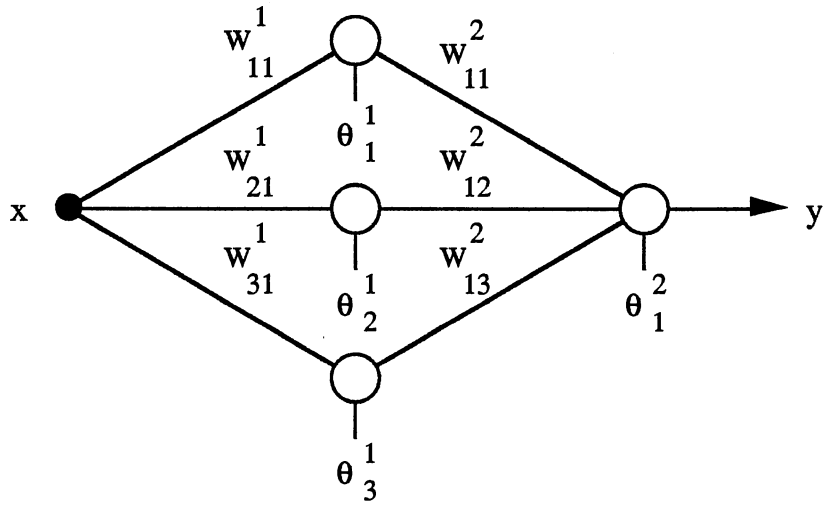


Figure 2.3. Single Input/Single Output Two Layer Perceptron

The weights shown below are for a two layer perceptron with one input, one output and 3 neurons in the first layer. This network is shown in Figure 2.3. Changes in the value of the weights and the offsets cause variations in the form of the function g: x-> y.

Figure 2.4 shows the response of the perceptron as these changes occur.

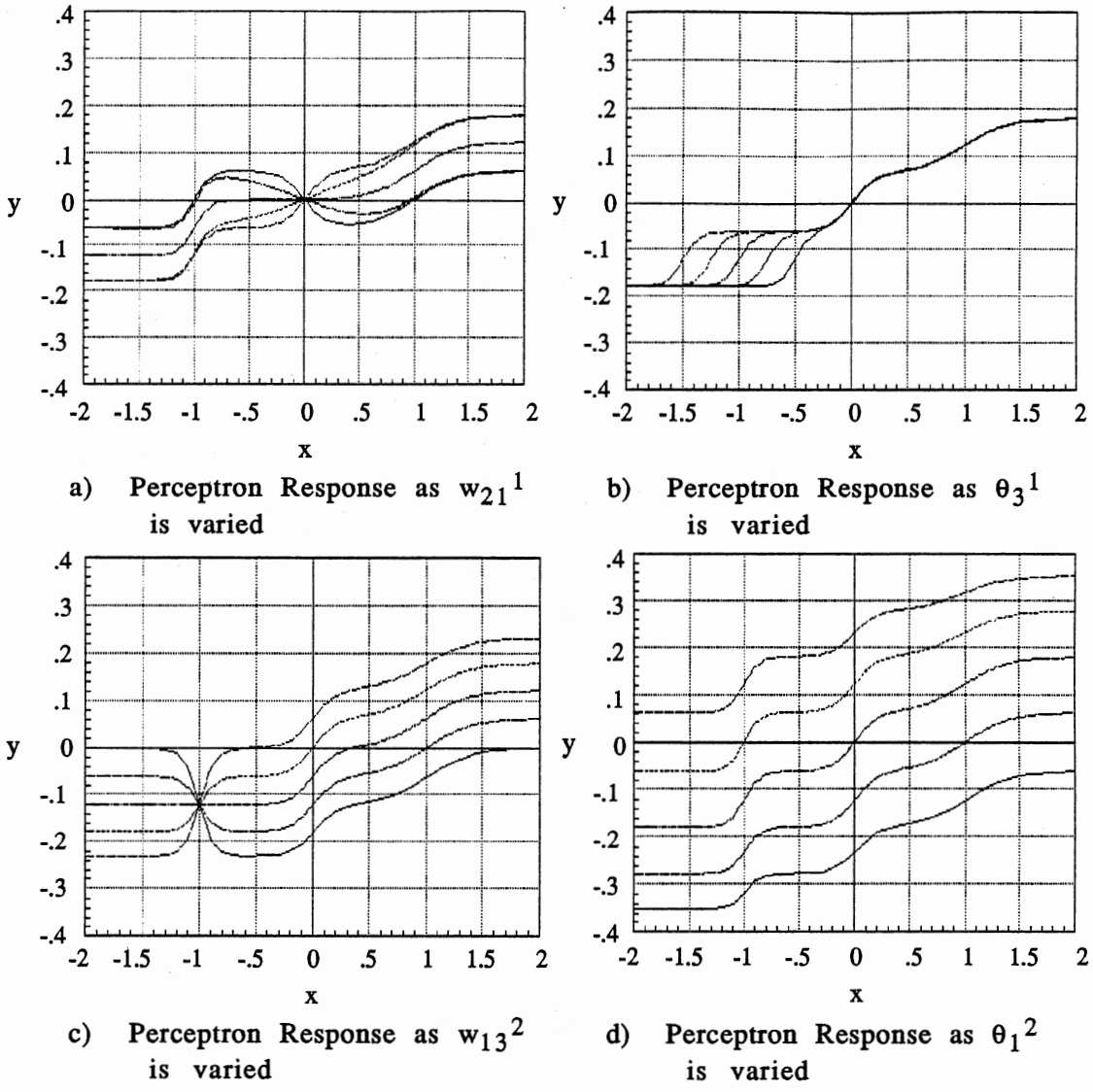
$$
\begin{array}{llll}
W_{11}^1 = 5 & \theta_1^1 = -5 & W_{11}^2 = .5 & \\
W_{21}^1 = 10 & \theta_2^1 = 0 & W_{12}^2 = .5 & \theta_1^2 = 0 \\
W_{31}^1 = 15 & \theta_3^1 = 15 & W_{13}^2 = .5 &
\end{array}
$$

Each step in the curves is caused by one particular neuron of the perceptron. The slope of the step is not only determined by the weight connecting the input to the neuron but also by the weight connecting this neuron to the output neuron (e.g. $w_{11}{}^1$ and $w_{11}{}^2$ determine the slope caused by neuron 1). The center of the step is determined by the negative of the ratio of the offset and the input weight (e.g. the center of the curve determined by neuron 1 is -(-5/5) = 1).

Figure 2.4 a) shows how the function changes as the value of $w_{21}{}^1$ is varied. This weight connects the input to the second neuron in the first layer. It takes on the values {-10, -3, 0, 3, 10}. As one can see, changes in this weight affect the slope of the middle step.

Figure 2.4. b) illustrates the function as the offset $\theta_3{}^1$ of the third neuron varies : {7.5, 11.25, 15, 18.75, 22.5}. Changes in this offset shift the center of the first step, compressing or expanding the function in the x axis.

In Figure 2.4 c) changes in the weight of the second layer can be seen. The weight $w_{13}{}^2$ is varied over the range {-1, -0.5, 0, 0.5, 1}. This weight affects the slope of the first step of the function and also shifts the entire function up or down.



a) Perceptron Response as $w_{21}{}^1$ is varied

b) Perceptron Response as $\theta_3{}^1$ is varied

c) Perceptron Response as $w_{13}{}^2$ is varied

d) Perceptron Response as $\theta_1{}^2$ is varied

Figure 2.4. Response of Perceptron in Figure 2.3

Finally, Figure 2.4 d) shows the effect of the offset of the second layer neuron, $\theta_1{}^2$. The entire curve is shifted up or down as this weight is varied from {-1, -0.5, 0, 0.5, 1}.

Figure 2.5 illustrates changes in the weight $w_{11}{}^1$. The values that this weight takes are {-10,-5,0,5,10}. Since this weight affects

the center of the step, as explained above, when it is negative the center of the step is shifted to the other side of the y axis. The center of neurons 1 and 3 then coincide and their efforts are added to shape the first step of the function. This exemplifies how small changes in only one weight can produce complex changes in the form of the function.

Figures 2.4 and 2.5 illustrate that the multilayer perceptron can approximate very complex functions. Although a two layer network can approximate any function, three or even four layers may be needed to reduce the number of neurons in the network. Determining the number of neurons and the number of layers needed to approximate a given function is difficult and is generally done by trial and error. The development of analytical procedures for this task is an active area of research.

Once a network structure has been chosen, the weight values which provide the closest approximation to the desired function must be determined. The most popular method for computing the optimal weights is the backpropagation algorithm [4], which is discussed in the next section.

The Backpropagation Algorithm

The backpropagation algorithm is widely used in the determination of the strength of the connections in the multilayer perceptron. The procedure minimizes the squared error obtained from the difference between the output of the network (o(x)) and the desired output (yd) summed over all training pairs. The

minimum is obtained by applying steepest descent to the following error measure

$$E = \frac{1}{2} \sum_{x \in U} (yd - o(x))^T (yd - o(x)) \tag{II.2}$$

where o(x) is the output of the network for input x and yd is the corresponding desired output. The error measure goes to zero as the network approximates the function yd more closely.

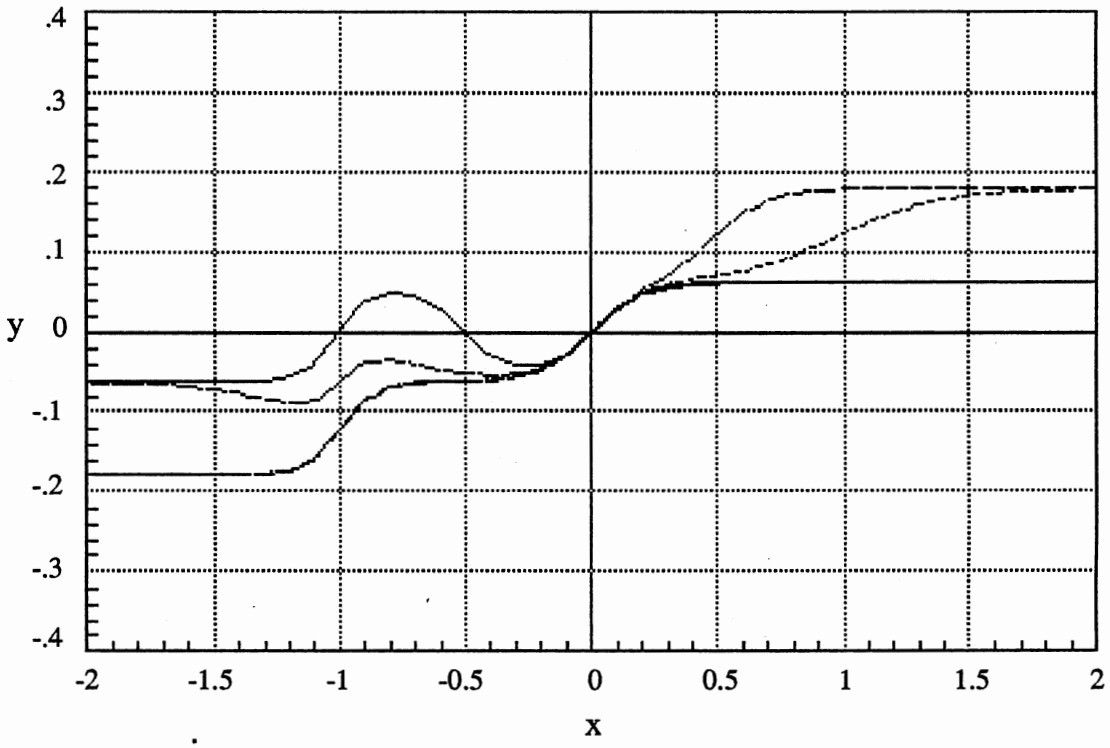


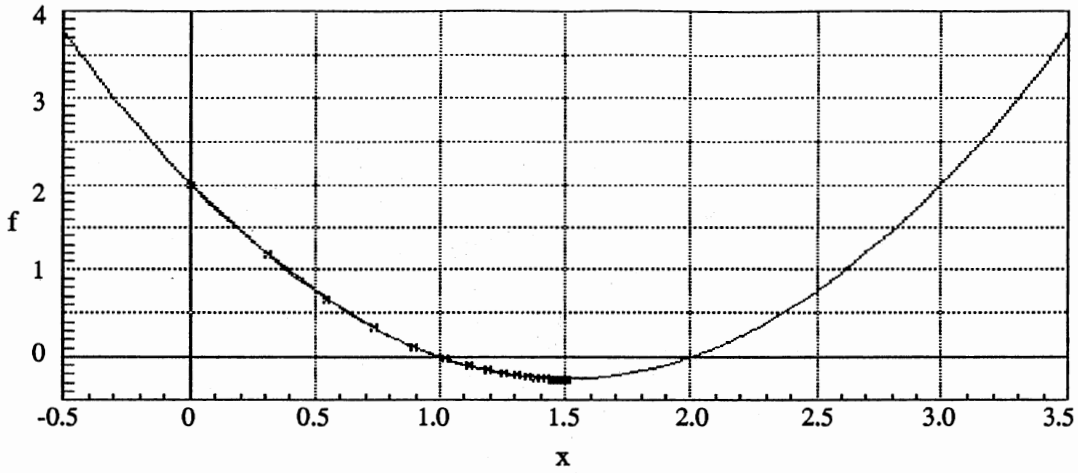
Figure 2.5. Response of Perceptron in Figure 2.3 as w_{11}^1 is Varied

The method of steepest descent [5] uses only first derivatives, the gradient of the error measure. The gradient is a vector that points in the direction of growth of the function. If the direction opposite to the gradient (i.e. the steepest descent direction) is taken, then the path used will decrease the function.

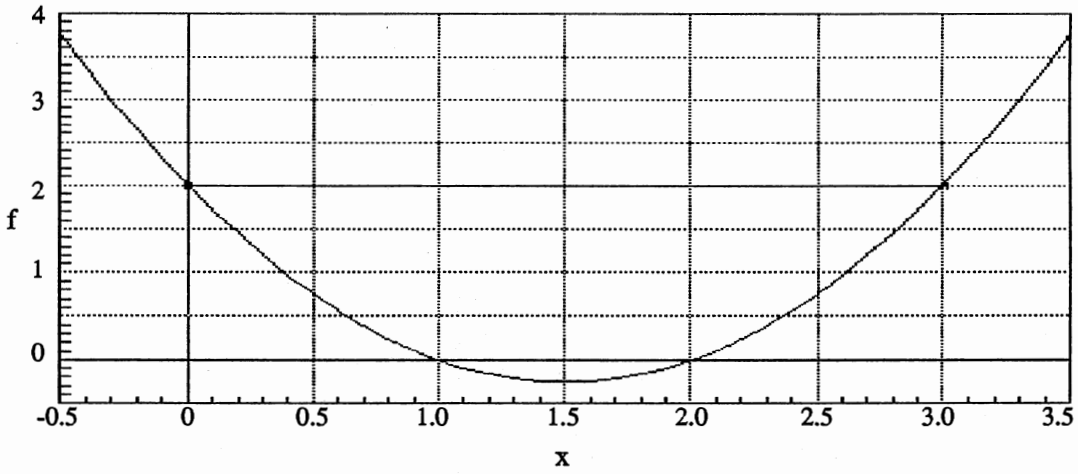
The gradient gives the direction that decreases the function but does not give the magnitude of the step to be taken in that direction. The methods used to calculate the magnitude of the step distinguish the various procedures of steepest descent. Many steps must be taken in the direction of steepest descent, since one step may not be sufficient to reach the minimum of the function.

Different methods can be used to select the step size. One of them is to use a fixed step size. If the step size is too small many iterations may be necessary to reach the minimum. If the step size is too large, oscillations around the minimum point may occur. Thus, it is better to use a variable step size. Large steps are taken in the first iterations. At each iteration the step size is reduced so that when the minimum is approached the step size is small.

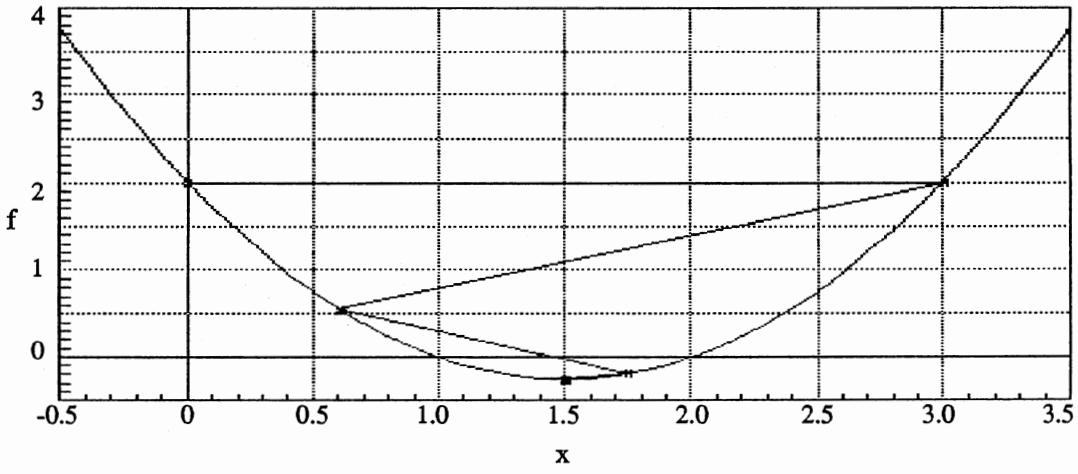
Figure 2.6 shows the quadratic function $f(x) = x^2 - 3*x + 2$ when variable and fixed step sizes are used to search for the minimum. In Figure 2.6 a) the step size is fixed at 0.1, a small value. The minimum is reached in 49 iterations of the algorithm. A large step size of 1.0 is used in Figure 2.6 b). The minimum is never reached. Oscillations occur between the values of $x = \{0, 3\}$.



a) Fixed Step Size at 0.1



b) Fixed Step Size at 1.0



c) Variable Step Size

Figure 2.6. Minimum Search in a Quadratic Function

Finally, Figure 2.6 c) shows the effect of using a variable step size. Only 12 iterations of the algorithm are necessary to reach the minimum. The initial value of the step size was 1.0, while the final value was 0.086. At each iteration the value of the step size was reduced by 20%.

The backpropagation algorithm is a steepest descent algorithm which is unique in the manner in which it computes the gradient. The training process consists of presenting the network with inputs $x \in U$ which are propagated forward through the network to produce an output o(x) at the last layer. This output is compared to desired outputs yd, obtaining an error whose derivative is propagated backwards through the network in order to compute the gradient. This process is repeated until the errors are small.

The following equations are part of one iteration of the backpropagation algorithm. Equation II.3 determines the output of each neuron

$$o_j = f(\sum w_{ji}\, o_i + \theta_j) \tag{II.3}$$

where o_j is the output of neuron j, θ_j is the offset of neuron j, w_{ji} is the weight connecting neuron i to neuron j, and f is the nonlinearity. The nonlinearity used here is the logistic function shown in Figure 2.2 and expressed here again

$$f(x) = \frac{1}{1+e^{-x}} - \frac{1}{2} \tag{II.4}$$

The weights are adjusted by the formula

$$\Delta W_{ji}(n+1) = \eta\, \delta_j\, o_i + \alpha\, \Delta W_{ji}(n) \tag{II.5}$$

where n denotes an iteration of the training algorithm, η is the learning rate, δ_j is the error signal of neuron j, $\delta_j o_i$ is the gradient and α is the momentum factor.

The momentum factor (α), together with the variable step size, helps reduce the number of iterations in the search for the minimum of the function, without introducing oscillations. It determines the percentage of past weight changes that will be incorporated in the new weight changes. It is similar to a low pass filter. If the error surface has a sharp curvature this may cause divergent oscillations in narrow valleys, so small step sizes are necessary. Furthermore, small step sizes lead to slow convergence, as shown above. Therefore, the momentum factor effectively increases the step size by filtering out these sharp curvatures.

The error signal for an output neuron is calculated as

$$\delta_j = (yd_j - o_j)\ f'(o_j) \qquad \text{(II.6)}$$

where yd_j is the desired output for neuron j and $f'(o_j)$ is the derivative of the nonlinearity (logistic function) which is

$$f'(x) = (0.5 + f(x))*(0.5 - f(x)) \qquad \text{(II.7)}$$

For neurons which are not output neurons, the error signal depends on the error signal of neurons that it connects to. The derivative is then backpropagated according to the following formula

$$\delta_j = f'(o_j)\sum_k \delta_k W_{kj} \qquad \text{(II.8)}$$

Figure 2.7 shows the learning behavior using the backpropagation algorithm of a two layer network with one input, one output and 12 neurons in the first layer. The function being

approximated is a sine wave. The erratic learning occurs because of the randomness of the inputs presented to the network as the learning process proceeds. As one can see, the network's learning is slow. Around 60,000 iterations were necessary for the network to reach an error of 0.001.

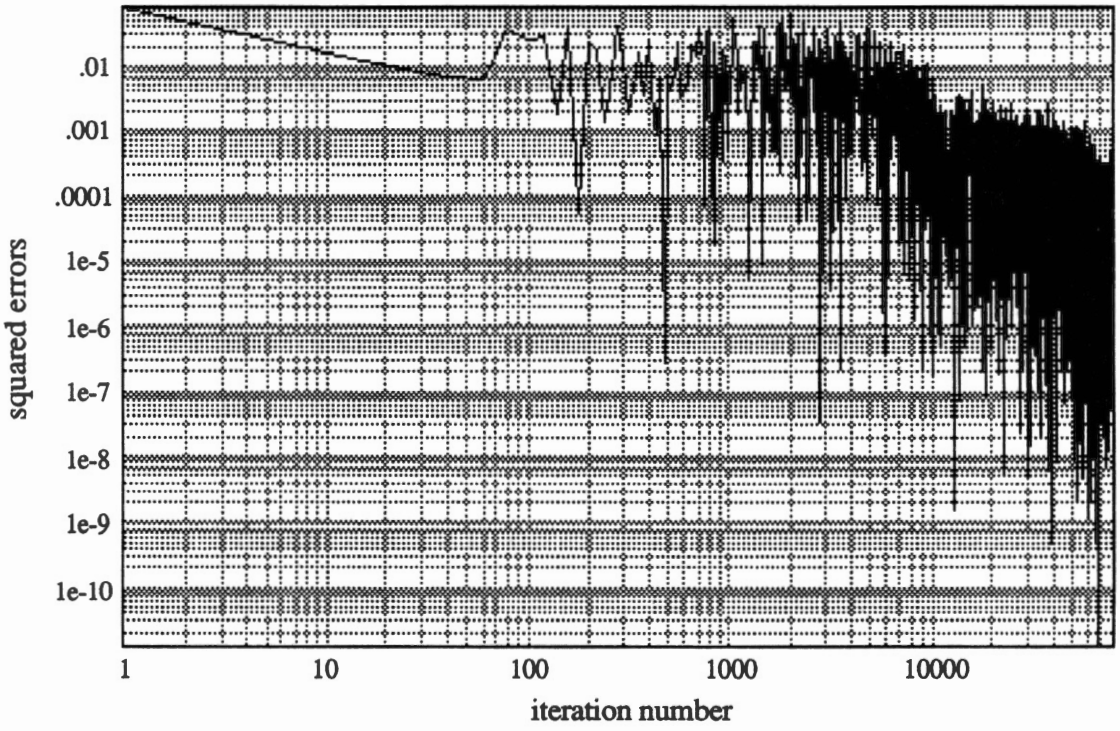


Figure 2.7. Network Learning Behavior for Standard Backpropagation

The slow convergence of backpropagation occurs not only because of the randomness of the inputs but also because of the shape of the error surface, as will be illustrated next.

Figure 2.8 shows the architecture of a 2 layer network along with its input/output characteristic. The following are the nominal weight values of the network

$$W_{11}^1 = 10 \quad \theta_1^1 = -5 \quad W_{11}^2 = 1$$
$$W_{21}^1 = 10 \quad \theta_2^1 = 5 \quad W_{12}^2 = 1 \qquad \theta_1^2 = -1$$

The nonlinearity used is

$$f(x) = \frac{1}{1+e^{-x}} \tag{II.9}$$

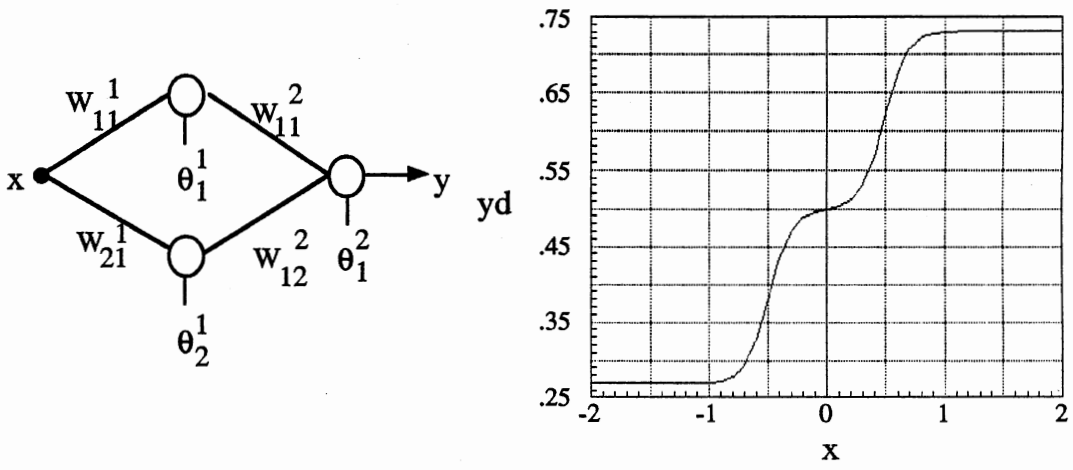


Figure 2.8. Single Input/Single Output Two Layer Perceptron with its Input/Output Characteristic

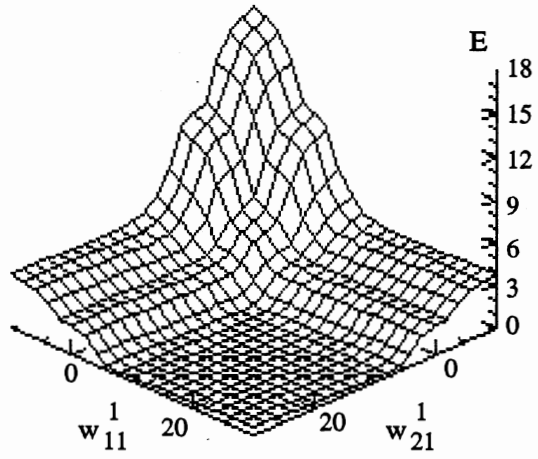
An experiment was performed to observe how backpropagation is affected by initial condition values. For this experiment the desired output is the output of the nominal network, shown in Figure 2.8. In order for the error surface to be displayed graphically, only two weights were allowed to change at one time. Figure 2.9 shows the different error surfaces obtained as different weights were selected to vary.

Figure 2.9 a) shows the error surface as weights w_{11}^1 and w_{21}^1 are varied. Figure 2.9 b) has offsets θ_1^1 and θ_2^1 varying. Figure 2.9 c) varies offset θ_1^1 and weight w_{11}^1. While in Figure 2.9 d) weights w_{11}^1 and w_{11}^2 are varying.

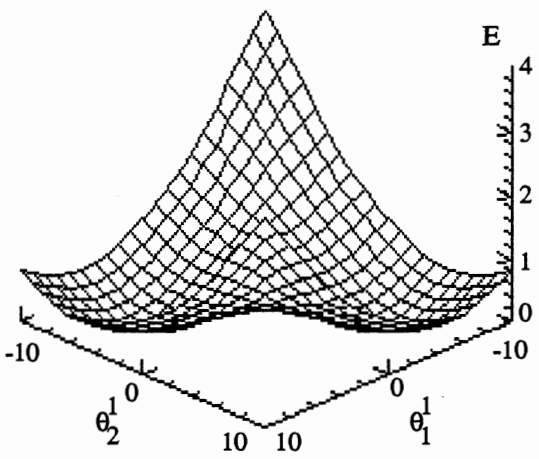
As different initial conditions are selected, convergence to different final conditions occur, except for the case shown in Figure 2.9 a), in which convergence to the global minimum occurs no matter what values are chosen for the initial conditions.

The case shown in Figure 2.9 b) is illustrated in Figure 2.10. Convergence to three different points can occur. Two of these points, (5,-5) and (-5,5), result in the same network because of symmetry. The third point is a saddle point at (0,0) which only happens when the initial conditions are exactly identical. This is rare since initial conditions are normally random.

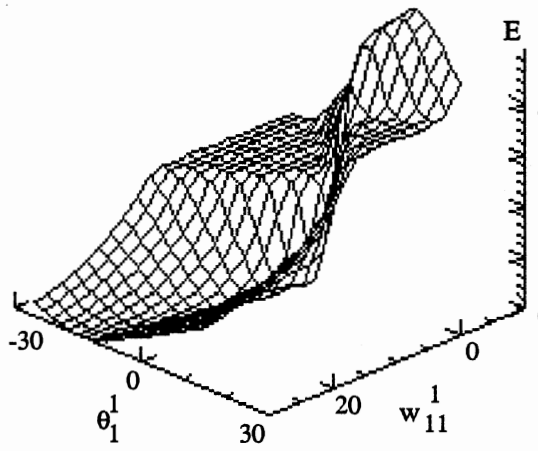
Figure 2.11 shows convergence to a global minimum of the error surface of Figure 2.9 c). If the initial parameter values fall in the area of the plateau surrounding point (0,-15), the backpropagation algorithm will not converge to a solution. This plateau is very shallow so its derivative is very small, affecting the performance of the backpropagation algorithm.



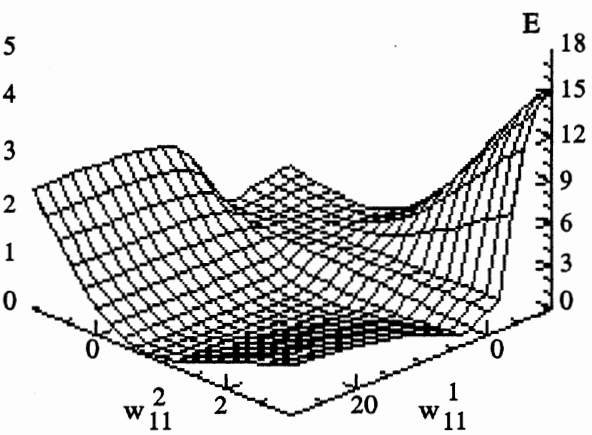
a) Error surface as $w_{11}{}^1$ and $w_{21}{}^1$ is varied



b) Error surface as $\theta_1{}^1$ and $\theta_2{}^1$ is varied



c) Error surface as $\theta_1{}^1$ and $w_{11}{}^1$ is varied



d) Error surface as $w_{11}{}^1$ and $w_{11}{}^2$ is varied

Figure 2.9. Error Surfaces for Network in Figure 2.8

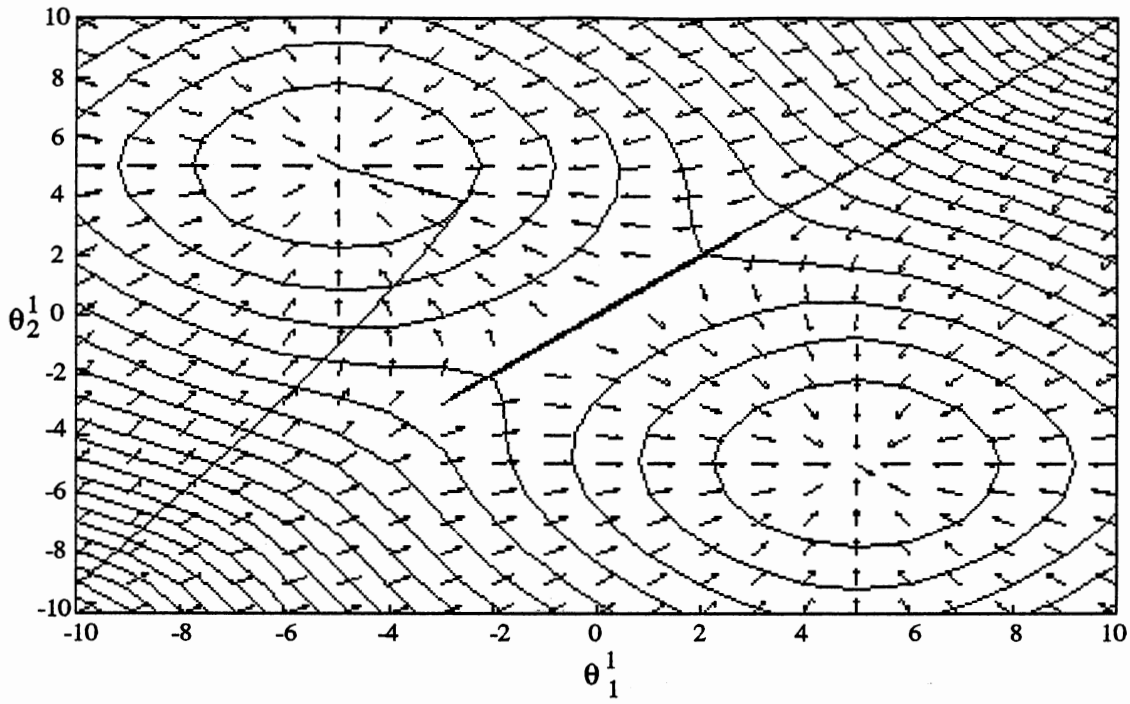


Figure 2.10. $\theta_1{}^1$ and $\theta_2{}^1$ Convergence for Different Initial Conditions

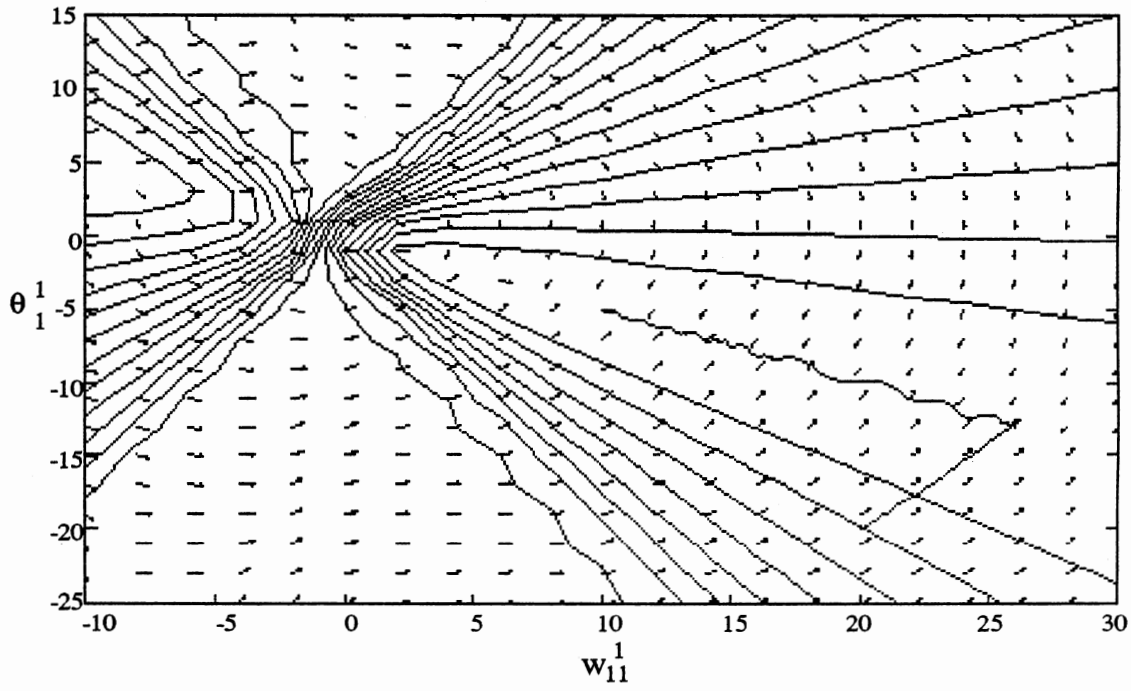


Figure 2.11. $\theta_1{}^1$ and $w_{11}{}^1$ Convergence to Global Minimum

Figure 2.12 shows case of Figure 2.9 d) for different initial conditions. One initial condition converges to the nominal value of the weight $w_{11}^1 = 10$ and $w_{11}^2 = 1$, but the other drifts away. This error surface shows two valleys, one local minimum where the second initial condition converges to and one global minimum - the nominal weight value. Error surfaces can look even more complicated than these and convergence to the global minimum is not guaranteed.

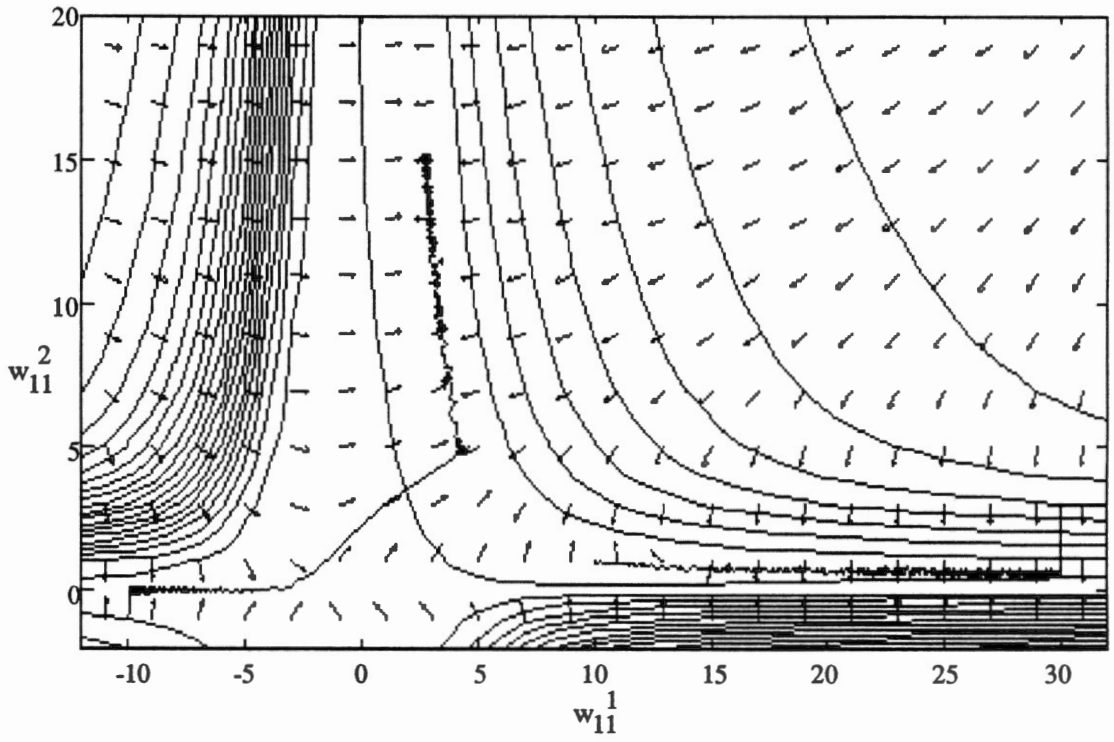


Figure 2.12. w_{11}^1 and w_{11}^2 Convergence for Different Initial Conditions

The backpropagation algorithm is characterized by slow convergence because of the complex error surfaces, and because of its gradient descent nature. In the next chapter a number of modifications suggested by other authors are described and analyzed [6].

CHAPTER III

MODIFICATIONS TO BACKPROPAGATION

A number of methods have been suggested to accelerate the backpropagation algorithm [6]. Some of these methods are investigated in this chapter.

The backpropagation training method consists of presenting at each iteration one input of the training set, obtaining the network's output and correcting the weight matrix based on this particular input.

The magnitude of the correction is proportional to the learning rate, η. A high η can result in faster convergence but it can also lead to oscillations. A small η will increase convergence time, but a decrease in the oscillations may occur. This sensitivity of η to the error surface is a major concern in selecting the value of η for the learning process. The error surface could be quite complex as was shown in the previous chapter.

Methods for Acceleration

Initially, four modifications to the standard backpropagation algorithm were considered. These four methods for acceleration are described below.

First, the weights are updated only after a certain number of inputs have been presented to the network [6]. Although changes to the weights are calculated after each input is presented to the network, the overall change is only applied after all inputs are presented. A sum of the changes is used to update the weight vector. **Loopmax** is the number of inputs used to generate this sum.

Second, the learning rate η is varied dynamically [6]. As explained above, η is very sensitive to the error surface. Thus, if a change in the weights results in a smaller error, the new weights are accepted and the value of η is increased by a factor ϕ. If the error increases, the new weights are rejected and the value of η is decreased by a factor of β. If the error increased but is below some percentage, the new weights are accepted and the value of η is unchanged. **Perover** is the term used here to define this percentage.

Third, the momentum factor α is also modified according to the dynamic variations of η [6]. The momentum factor weighs previous changes of the corrections. If the error decreases then the previous changes will aid convergence; the optimization direction is correct. But if the error increases, a change in optimization direction is necessary and α is set to zero.

Finally, the normalization of the initial weight values can also help to increase the convergence rate of backpropagation [7]. The weights are normalized such that each neuron is assigned an interval of the input range. This will be described later.

In the next section these modifications will be used when a sinusoidal function is to be approximated. The effects of the parameters α, **loopmax**, **perover**, β, ϕ, and the effect of the initial weight values are investigated. In addition, the effects of the number of neurons and the number of layers on the algorithm convergence and on network performance is analyzed. Also, the conjugate gradient method will be compared with modified backpropagation.

Case Studies

Next a series of experiments which were performed on the perceptron neural network, in combination with the backpropagation learning algorithm, are delineated. First, the basic test setup is described, then two sets of tests are discussed. The first set of tests investigates parameters which affect the convergence of the learning algorithm. The second set of tests studies the effect of the numbers of neurons and layers on network performance.

The purpose of these experiments is to gain additional insight into how algorithm parameters (e.g., the momentum factor, the learning rate) can be selected or modified to improve the performance of the network and/or convergence of the algorithm. For this study the performance of the network is measured by the sum of squared errors (see Equation II.2).

Basic Test Setup

The function to be approximated by the neural network in these experiments is a sine wave. The reason for selecting this function is two-fold: first, this nonlinear function is well known and results can directly be interpreted from Figure 2.4, and second, sinusoidal functions are typical of robotic applications. The exact function is given by

$$y = \frac{1}{2} + \frac{1}{4}\sin\left(\frac{\pi}{2}nx\right) \qquad -1 < x < 1 \qquad \text{(III.1)}$$

where: x - input

y - output

n - frequency parameter

The parameter n adjusts the frequency of the sine wave and can be used to investigate certain properties and capabilities of the neural network under study.

The backpropagation algorithm is known for its slow convergence time and the large computational power it requires. For this study the algorithm was implemented on the Intel iPSC/2 5-dimensional hypercube computer, a distributed memory (loosely coupled) MIMD machine. This required that the algorithm be parallelized. The basic idea is to equally divide the number of inputs over the hypercube and have each node calculate the corresponding errors. These errors are sent to the root node where a sum is calculated, and weight updates are determined and broadcast back to all other nodes. The weights on all the nodes are identical.

The inputs presented to the network are random numbers uniformly distributed in the interval selected by the user (in the examples below the chosen interval was [-1,1]). A total of **loopmax** inputs are presented to the network before the errors are summed and the weight updates are calculated.

Convergence Parameters

It was shown above that several parameters influence the learning algorithm convergence. The effects of these parameters are described next.

The network used in the following tests is a two-layer perceptron with one input, one output and 12 neurons in the first layer, i.e. a 1-12-1 configuration. The following parameters were fixed to the values indicated, unless otherwise specified:

α = .99

loopmax = 320

perover = 1.15

β = 0.9

ϕ = 1.11

The first parameter tested was the momentum factor, α. This parameter controls the percentage of the previous change in the weights that shall be incorporated in the new weight values (see Equation II.5). The values tested were: 0.25, 0.5 and 0.99. Figure 3.1 shows the performance of the network when α changes. The highest α gives the fastest convergence time, about 200 iterations. Even though α almost doubles from one value to the next, the

convergence rate does not improve in the same ratio. Also observe that higher values of α result in less noise in the sum of squared errors.

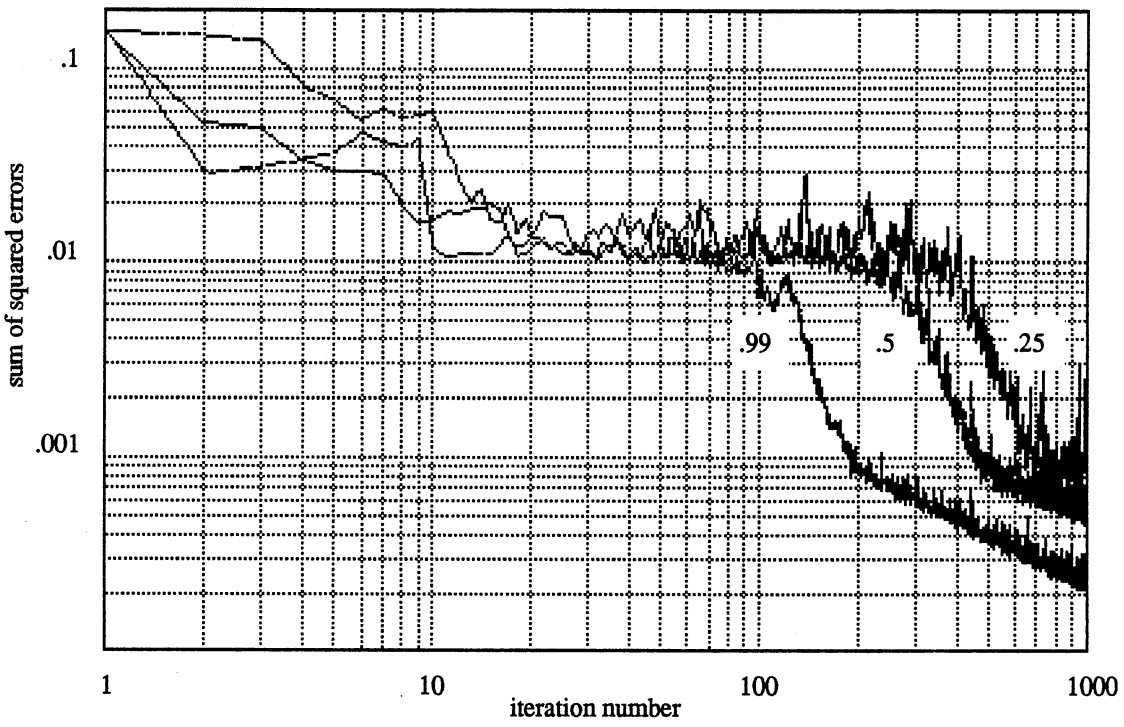


Figure 3.1. Network Learning Behavior as α is Varied

Figure 3.1 might suggest that even larger values for α might improve performance even more. Larger values were tested, however, and 0.99 was found to be optimal for this experiment.

The second parameter tested was **loopmax** - the number of inputs that are presented to the network before corrections are made to the weights. By altering **loopmax** the performance of the algorithm is modified, as shown in Figure 3.2. The numbers examined were: 80, 160 and 320. There was nothing special about these numbers, except for the fact that they are exactly divisible by the number of nodes of the hypercube, 16. The minimum convergence time was 200 iterations, given by the maximum value of **loopmax**. With this value less noise was introduced in the sum of squared errors. This was expected since the algorithm is effectively averaging over more inputs. One can also notice that the effect of doubling **loopmax** does not improve the performance in the same amount. Higher values for **loopmax** were also tested, and it was found that 320 was optimal for this experiment.

The next parameter tested was **perover**. This parameter regulates when α, the momentum factor, is set to zero, as described above. It also controls when to reject the new set of weight values. Figure 3.3 shows the performance of the network when **perover** is 1.01, 1.05 and 1.15. The highest value of **perover** gave faster convergence time, about 200 iterations. However, as **perover** is increased further more noise is found in the sum of squared errors. Therefore, the effect, with respect to noise, of increasing **perover** is the opposite of that in increasing α or **loopmax**, i.e. the noise increases while in the others the noise decreases. Values of **perover** larger than 1.15 were tested but did not improve performance.

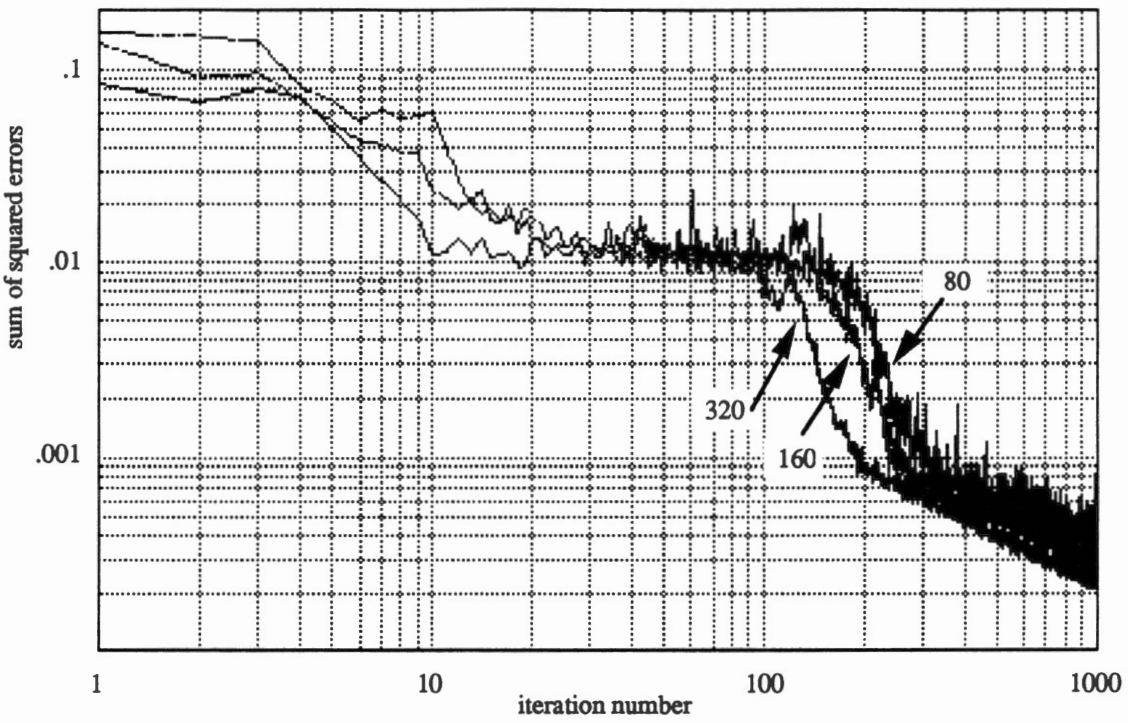


Figure 3.2. Network Learning Behavior as **loopmax** is Varied

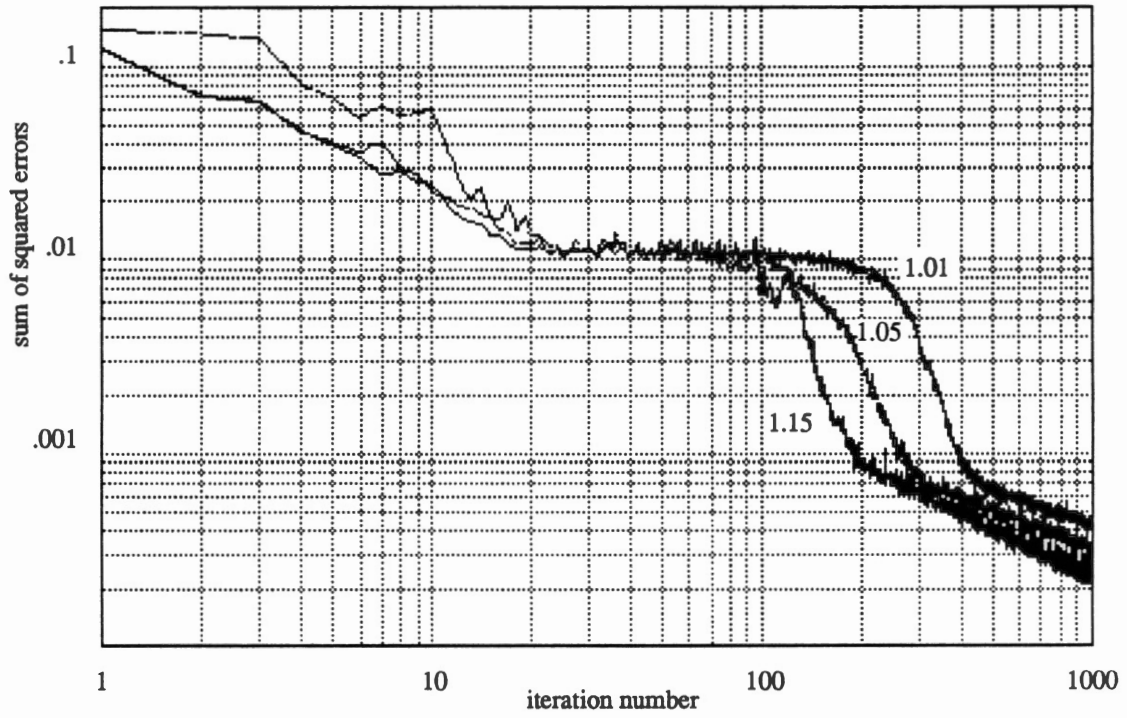


Figure 3.3. Network Learning Behavior as **perover** is Varied

The next parameter tested was β, the factor by which η is decreased when performance is poor. The values tested were: 0.8, 0.9 and 0.99. Figure 3.4 illustrates the performance of the network as β varies. The best performance was obtained for $\beta = 0.8$. Other values of β were tested but convergence was not improved.

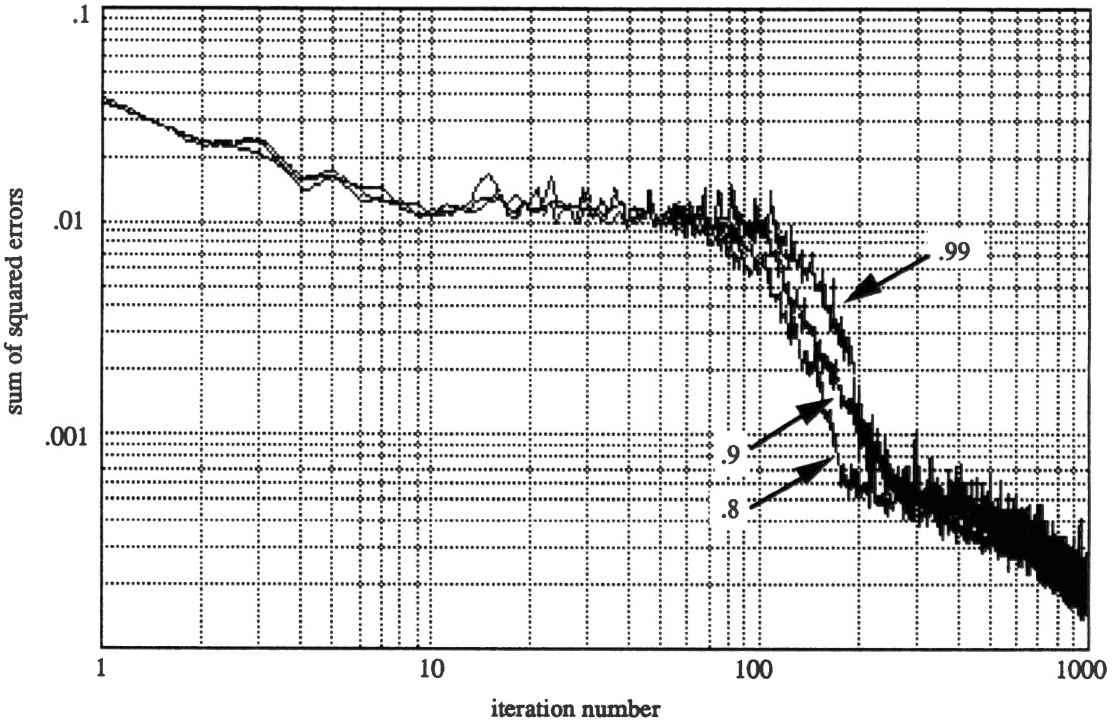


Figure 3.4. Network Learning Behavior as β is Varied

When performance is good the parameter ϕ increases the value of η. Figure 3.5 shows how the performance of the network changes for values of ϕ of 1.05, 1.11 and 1.20. The value of 1.05

gave faster convergence. Other values of ϕ were tested and 1.05 was found to produce best results.

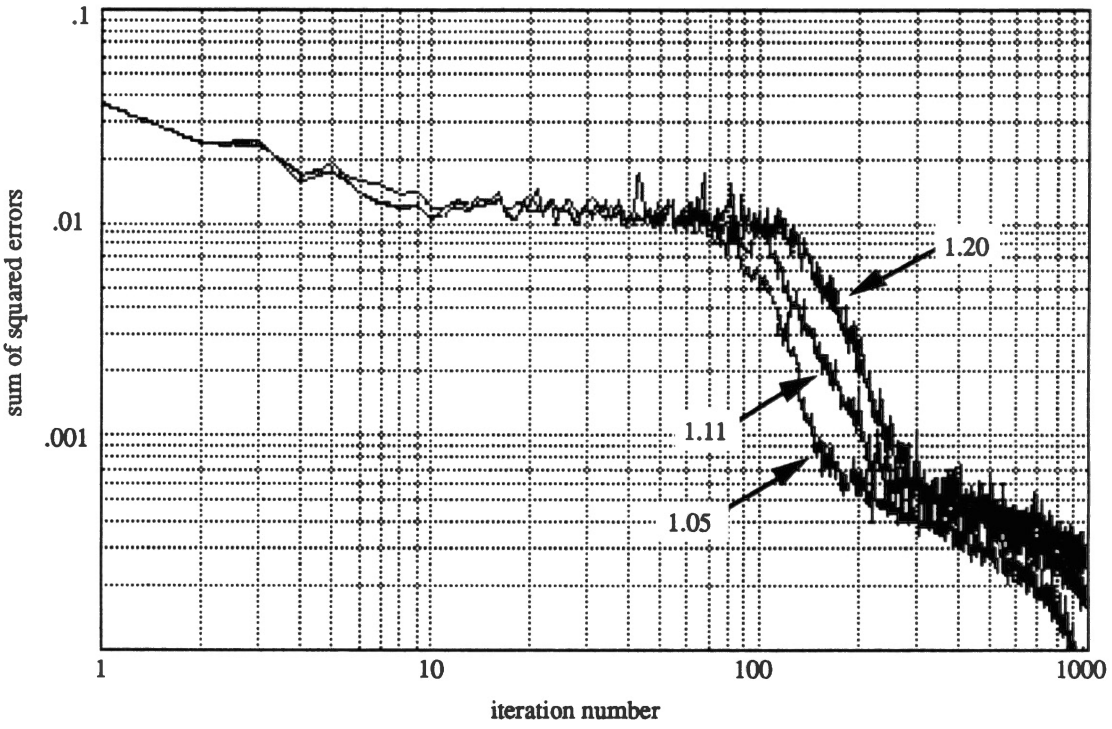


Figure 3.5. Network Learning Behavior as ϕ is Varied

The Numbers of Neurons and Layers

The above tests help in better choosing the main parameters for training the network. With these parameters set appropriately, the next step is to see how the frequency of the sine wave (the last parameter) affects the performance of a particular network. The

configuration used was a two layer network with one input, one output and three neurons in the first layer, i.e. a 1-3-1 network.

Figure 3.6 shows the approximation given by the neural network after it was trained for 10000 iterations when different frequencies are selected for the sine wave. The frequencies chosen were multiples of π (0.5π, π, 2π and 4π) giving the frequency parameter, n, the values: 1, 2, 4 and 8 respectively. As can be seen for n equal to 1 and 2, the approximation matches the desired sine wave very well. For n equal to 4, the network is reaching its maximum capability. As was discussed above (see Figure 2.4), with a 1-3-1 network a limited number of inflection points can be obtained. Therefore, for n equal to 8, the figure shows that the network can only match a part of the curve. Even though there are many other solutions which the algorithm might have converged to, they would not match the desired sine wave better than the one presented here.

The 1-3-1 network could not approximate the higher frequency sine wave, unless more neurons were added in the second layer. Next a case will be shown where a 1-3-1 network does not converge, but a network with more neurons does converge. The network used was a two layer network with one input, one output and 2, 3, 4 or 5 neurons in the first layer, i.e. configurations 1-2-1, 1-3-1, 1-4-1 and 1-5-1 respectively. Figure 3.7 shows the approximation given by the networks 1-2-1, 1-3-1, 1-4-1 and 1-5-1 after they were trained for 10000 iterations for a fixed frequency.

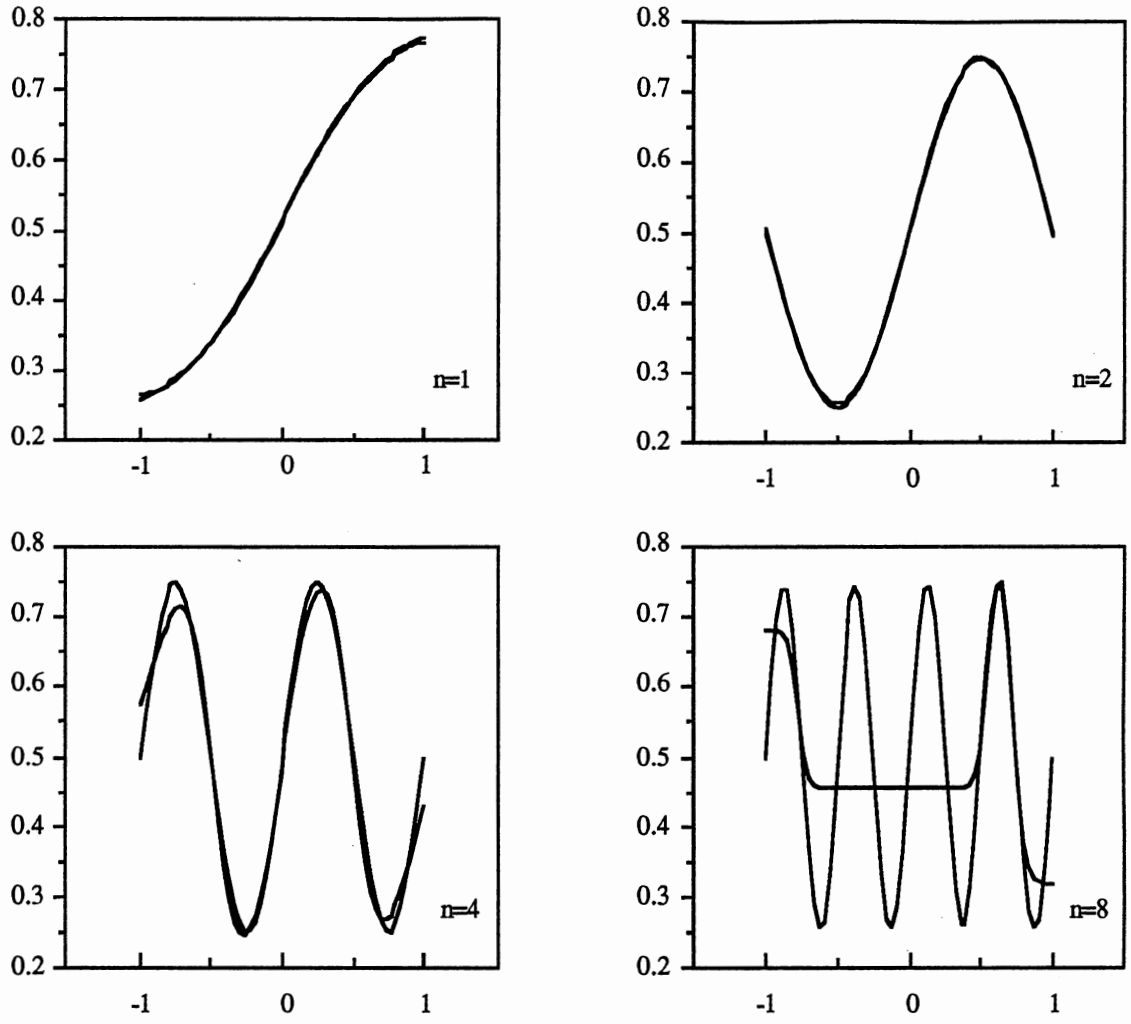


Figure 3.6. Mapping Capabilities of a 1-3-1 Network

The 1-2-1 network has a maximum of 3 inflection points, and it cannot match the desired sine wave. As the number of neurons, and therefore the number of inflection points, increases the approximation improves. Therefore, if the frequency increases, the number of neurons must also increase, to achieve an accurate approximation.

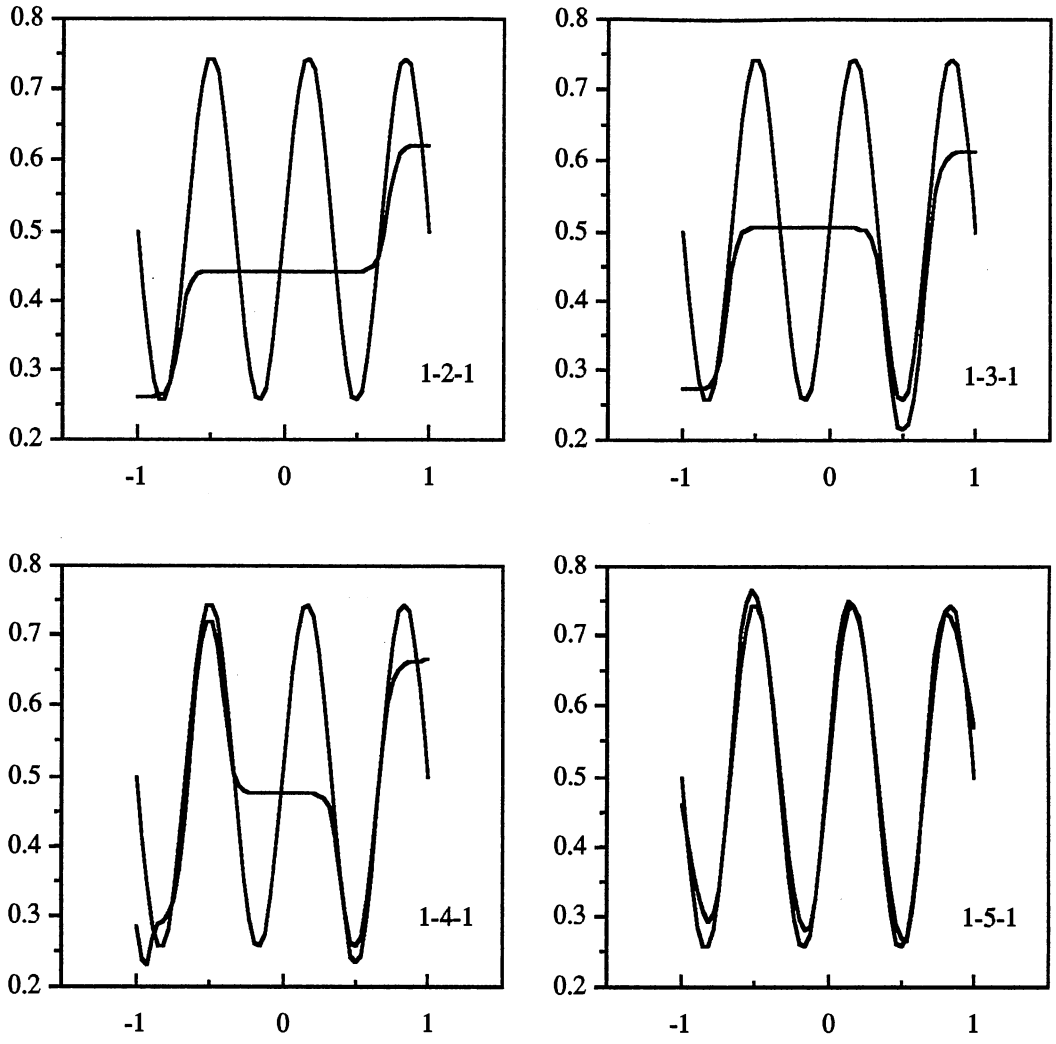


Figure 3.7. Network Mapping Capabilities

So far all of the cases shown here converged to the best possible performance of the network. Hecht-Nielson [3] proves that a two layer network can approximate any square integrable function, if there are enough neurons. But, he also mentions that sometimes the necessary weights cannot be determined through any training algorithm. The following example will illustrate the

sensitivity of the backpropagation algorithm. One network will be tested with two different choices for the initial weights. In one case the algorithm will converge to an optimum solution. In the other case the algorithm will converge to a suboptimum solution.

Initial Conditions

In this example, the network was a three-layer network with one input, one output, six neurons in the first layer and three neurons in the second layer (i.e., a 1-6-3-1 configuration). Two different random initial weights are given to the algorithm, and the results of the convergence process are shown in Figures 3.8 and 3.9. The mappings given by the initial weights are labeled 0. Succeeding numbers indicate network mappings in later iterations. In each case, the curve labeled 4 is the result of 2000 iterations. A total of 10000 iterations was performed, but little further change was seen in the shape of the functions (although, the case illustrated in Figure 3.8 did come to match the sine wave exactly).

As one can see from these figures, the mappings given by the different initial weights evolved to very different final results. It was found that as the numbers of layers and neurons increased, the likelihood of convergence to a local minimum also increased significantly. In some cases it was necessary to try many different initial conditions before a satisfactory result was obtained, especially if more than two layers were used. Even when a neural network of a specified architecture does exist, which can accurately

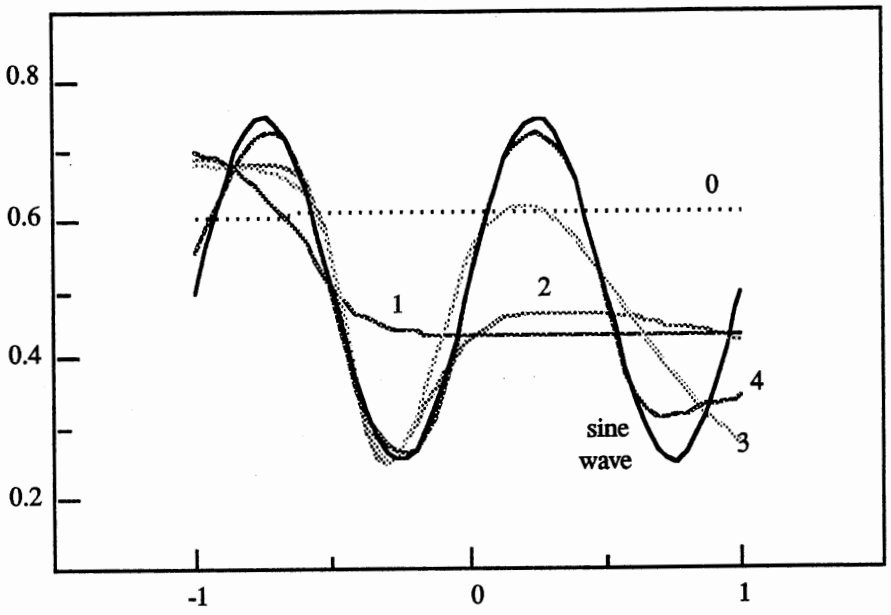


Figure 3.8. 1-6-3-1 Network Convergence to Global Minimum

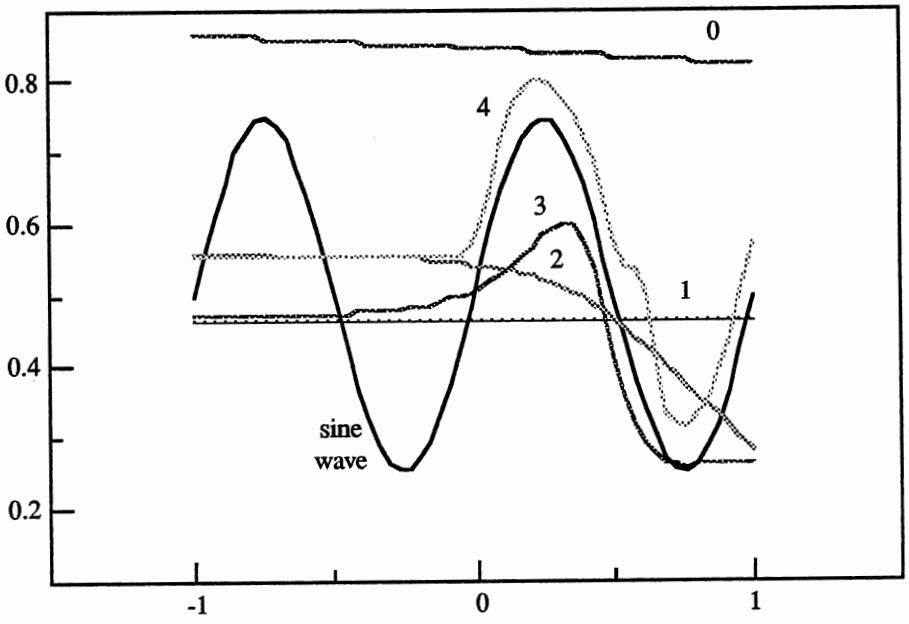


Figure 3.9. 1-6-3-1 Network Convergence to Local Minimum

approximate a given function, it may be difficult, or impossible, to determine the weights which can produce the correct mapping.

Initial Normalization

To reduce training time, normalization of the initial weight values is helpful [7]. The method consists of assigning overlapping intervals of the input range (active region) of the function being approximated to each hidden neuron in the network. This is done by first adjusting the magnitude of the weight vectors and then determining the offset so that the center of each sigmoid will fall in the active region.

For example, let the function being approximated be $g(x)=\sin(x)$ for $x \in [-1,1]$. The length of the active region is 2. If the network configuration is 1-2-1, then the number of inputs, n, is 1; the number of hidden units, κ, is 2 and the number of outputs is 1. The sigmoid for each hidden neuron is centered at $-\theta_i/w_i$ (θ_i - offset and w_i - weight from input to neuron i) and should be approximately linear over some subset of the active region. Since the active region should be distributed evenly among the hidden neurons, each subset should be of length $2/\kappa$.

The sigmoid is approximately linear in the interval -1 to 1. For the case of one input, the linear region for a neuron would be

$$-1 < w_i x + \theta_i < 1$$

or

$$(-1 - \theta_i)/w_i < x < (1 - \theta_i)/w_i$$

The length of this interval is $2/w_i$. Since each neuron should be linear over an interval of length $2/\kappa$ this would require $w_i = \kappa$. To

have overlapping intervals choose $w_i = 0.7\ \kappa$, for example. The intervals for each neuron are then spaced randomly throughout the active region by selecting each θ_i randomly in the interval $[-w_i, w_i]$.

In the case of n inputs, w_i and x are vectors of dimension n. The active region falls in a hypercube in $\Re^n$. In the case of only one input this hypercube was in $\Re$, a line.

The direction of each w_i determines the direction of the sigmoid for each neuron in the hidden layer. In the case of only one input all w_i's were in the same direction.

The magnitude of each w_i is proportional to the size of the interval in each direction. Since each input ranges from -1 to 1, the length of each interval would be 2/I, where I is the number of intervals per input. Let the magnitude of the weight vector of neuron i in layer k be expressed as

$$\left|w_i^k\right|^2 = \sum_j (w_{ij}^k)^2 \qquad \text{(III.2)}$$

The magnitude of the weight vector would be adjusted such that

$$\left|w_i^k\right| = I$$

Since there are n inputs, there are a total of I^n intervals in the hyperplane in $\Re^n$. Each hidden neuron is responsible for one interval so,

$$I^n = \kappa \qquad \text{or} \qquad I = \kappa^{1/n}$$

where κ is the number of hidden neurons. Therefore,

$$\left|w_i^k\right| = \kappa^{1/n} \qquad \text{(III.3)}$$

In the case of only one input, $w_i^k = \kappa$ and there are κ intervals in one direction, one for each hidden neuron. These sigmoids are then

distributed randomly in the hyperplane by choosing the offset as a random number in the interval $[-|w_i^k|,|w_i^k|]$.

Initially the weights are assigned random numbers from -1 to 1. The normalization is done by multiplying each weight w_{ij}^k by

$$\kappa^{1/n} \Big/ p|w_i^k| \qquad \text{(III.4)}$$

where p is the lower of the maximum of the n input values ($p=\min(\max(x_1),\max(x_2),\ldots,\max(x_n))$. Then the input range (active region) is distributed among the hidden neurons.

During training the weights are free to change according to the learning scheme presented above. Figure 3.10 shows the improved performance of the algorithm when normalization is done.

Conjugate Gradient

Another well known algorithm of steepest descent nature is the conjugate gradient method [5,8]. In this method steps are taken in conjugate directions instead of in the direction of steepest descent. The initial direction is selected arbitrarily, e.g. the gradient direction. The next direction is the conjugate of the previous direction, i.e.

$$p_n^T \nabla^2 E(w_o)\, p_o = 0$$

where: p_n - the new direction

p_o - the previous direction

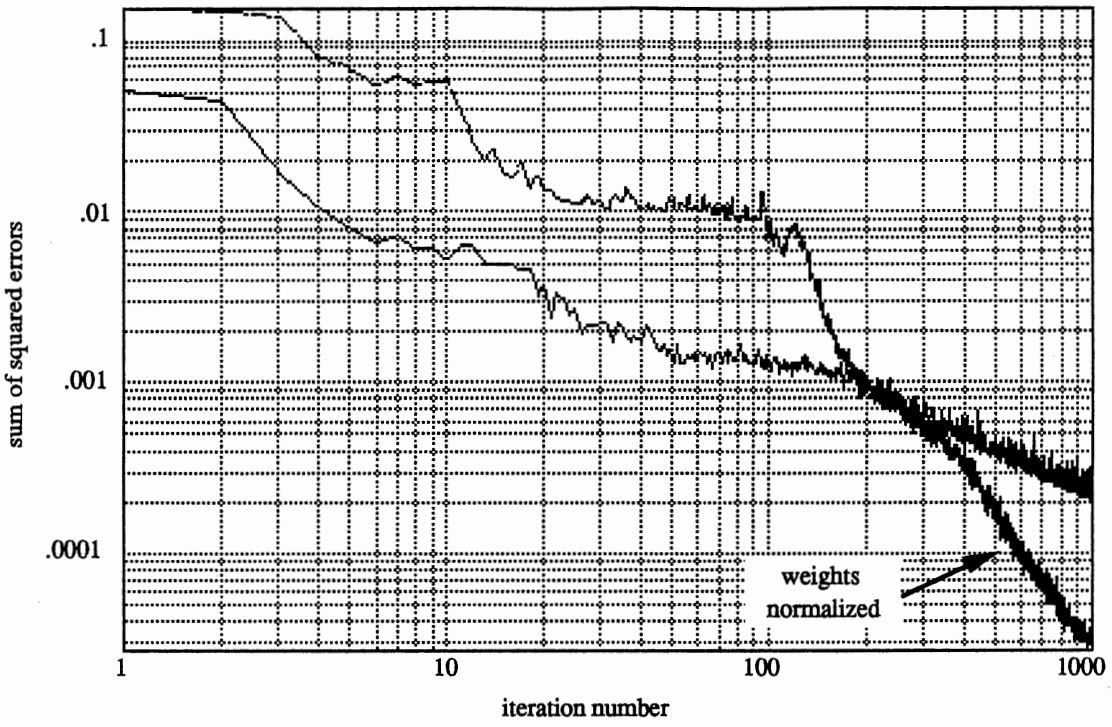


Figure 3.10. Network Learning Behavior with Weight Normalization

$\nabla^2 E(w_o)$ - the Hessian of the performance index evaluated at w_o

w_o - the weight values

One characteristic of conjugate gradient methods is the quadratic termination, i.e. if this method is used to minimize a quadratic objective function with n independent variables then only n steps are required to obtain the minimum if one exists.

One of the many conjugate gradient methods is Fletcher-Reeves [9]. A property of the Fletcher-Reeves conjugate gradient method is that the sequence of search directions are linear combinations of the current steepest descent direction and previous search

directions. The new search direction is computed by using only the current gradient direction and the gradient direction obtained in the previous iteration. So, the new search direction p_n is determined as

$$p_n = grad_n + p_o \frac{grad_n^T grad_n}{grad_o^T grad_o} \qquad \text{(III.5)}$$

where: $grad_n$ - current gradient direction

$grad_o$ - previous gradient direction

The gradient directions are obtained through backpropagation (see Equation II.5), but without using the momentum term.

At each iteration a search is performed to locate the new weight values, w_n, as the minimum of the performance index in that direction. The golden section search is performed here. First, an interval is obtained where a minimum of the performance index lies, say [a,b] where $a = w_o$ and $b = w_n$. Next, this interval is split into 3 segments. Points c and d are located on the interval between a and b such that $c = a + (1-\tau)(b-a)$ and $d = b - (1-\tau)(b-a)$ where $\tau \cong 0.62$. Finally, the next interval is computed as follows

if $E(c) < E(d)$ then

$b = d$; $d = c$; $c = a + (1-\tau)(b-a)$

else

$a = c$; $c = d$; $d = b - (1-\tau)(b-a)$

end

repeat until $b - a < \varepsilon$, where ε is a small constant.

This is illustrated in Figure 3.11.

The main property of this search is that the ratio of the whole interval to the larger segment is the same as the ratio of the larger segment to the smaller one. Also, the number of iterations need not be determined in advance, a termination criterion stops the search process at any iteration.

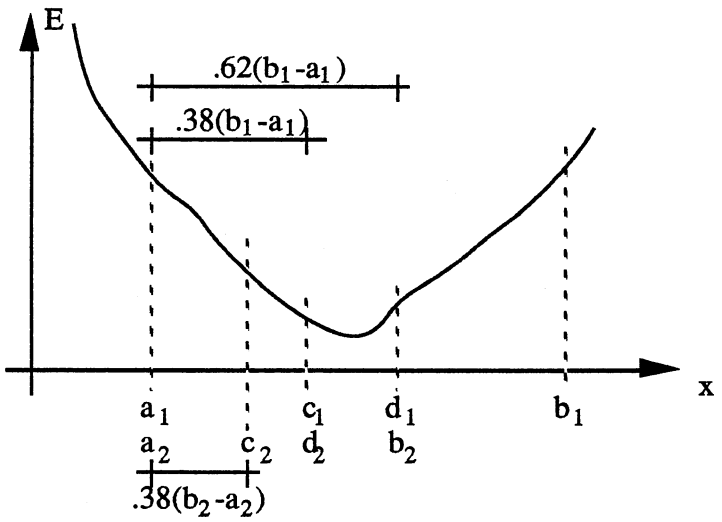


Figure 3.11. Golden Section Search

In Figure 3.12 the performance of the conjugate gradient is compared with that of modified backpropagation for the 1-12-1 network. Conjugate gradient takes more computing time than backpropagation for each iteration, but it takes fewer iterations to converge. Overall, it appears that the conjugate gradient method

often takes less computer time, on a serial computer, than does backpropagation, even when acceleration methods are employed.

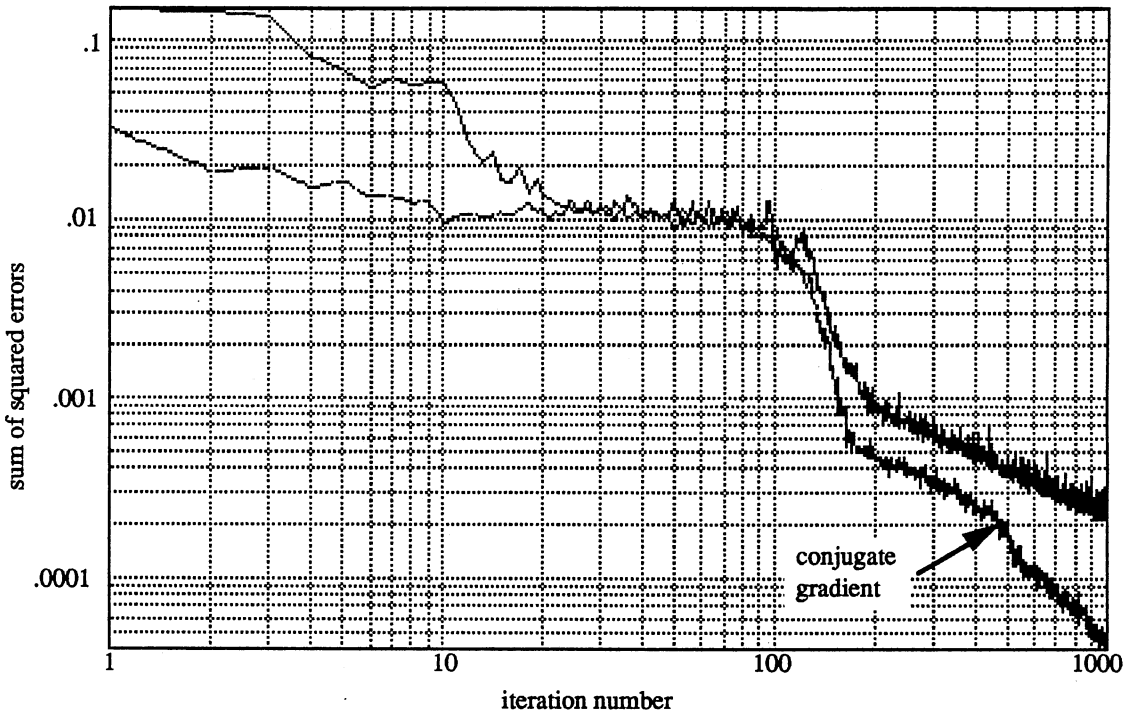


Figure 3.12. Network Learning Behavior for Conjugate Gradient Method

CHAPTER IV

CONTROL APPLICATIONS

The multilayer perceptron, with the backpropagation learning algorithm, is the most commonly proposed neural network architecture for control systems. This chapter will begin with a simple example of the use of a perceptron to approximate a linear control law. Following this introduction, three more complex techniques for implementing control laws with feedforward neural networks will be described.

Neural Networks in Control

Consider a system described by the following ordinary differential equation

$$\dot{x} = g(x,u) \tag{IV.1}$$

where x is the state of the system and u is the system input. The goal is to drive the plant from an initial state x_o to a final desired state x_d. Measurements of the states of the plant are available and are used to determine the control input according to the feedback equation

$$u = h(x) \tag{IV.2}$$

The first step in the standard design of a linear controller for a nonlinear system is the linearization of the plant around an

operating point x_o. Then, the controller is designed for the linearized model. This controller is finally used with the nonlinear system. Pole positioning is one of the many methods used in designing a controller [10].

Pole positioning consists of positioning the poles of the closed loop system at desired locations by allowing the feedback input to be a linear combination of the states x

$$u = -k*x \tag{IV.3}$$

If the linearized model of the system is

$$\dot{x} = A_L x + B_L u \tag{IV.4}$$

then the eigenvalues of the closed loop system (A_L - B_L*k) are the new pole locations.

An example of a simple nonlinear system is the inverted pendulum shown in Figure 4.1. The cart-pendulum system runs in a horizontal plane with friction and is controlled by applying to the cart the horizontal force u. There is no friction at the pivot point. The equations of the system are

$$\begin{aligned} M\ddot{s} &= u - \mu\dot{s} \\ \ddot{\phi} - \frac{g}{A}\sin\phi + \frac{1}{A}\ddot{s}\cos\phi &= 0 \end{aligned} \tag{IV.5}$$

where $A = \dfrac{J + mL^2}{mL}$, M - mass of the cart, μ - friction coefficient, L - distance from pivot to center of gravity of the pendulum, g - acceleration due to gravity, m - mass of the pendulum, u - force applied to the cart, J - moment of inertia of the pendulum, $\dot{s}$ - first derivative with respect to time, $\ddot{s}$ - second derivative with respect

to time. The goal is to balance the pendulum (ø=0) and take the cart to the origin (s=0).

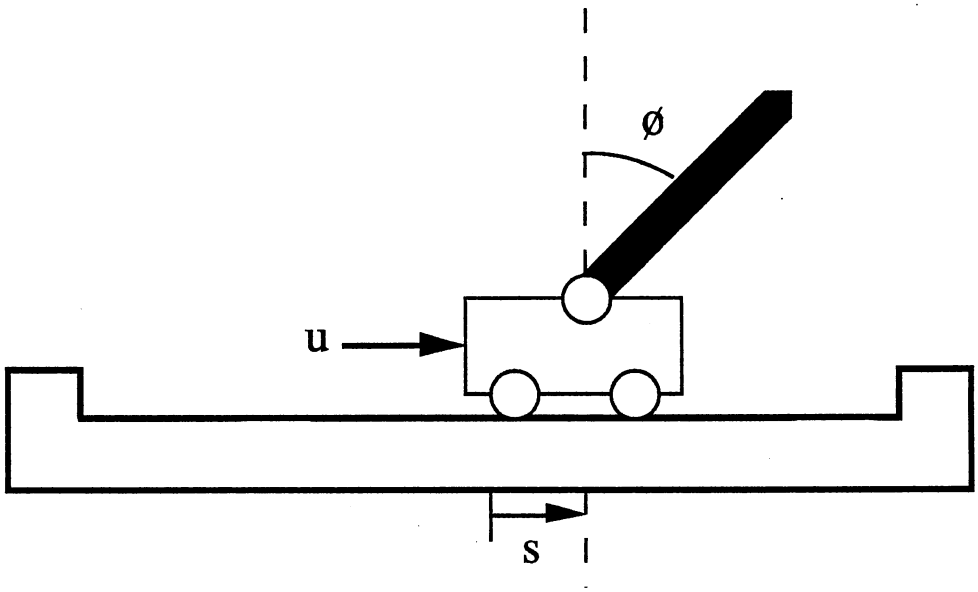


Figure 4.1. Inverted Pendulum System

Consider a linear controller given by

$$u = k_1 * \phi + k_2 * d\phi/dt + k_3 * s + k_4 * ds/dt \qquad \text{(IV.6)}$$

where k_1 through k_4 are constants. Figure 4.2 illustrates the system response for a particular linear controller (k_1 = 65.65, k_2 = 11, k_3 = -72.6, k_4 = -21.27) when the pendulum is initially tilted by 30 degrees. This controller was designed for the linearized system about ø=0 and such that all closed loop poles were at location -3.

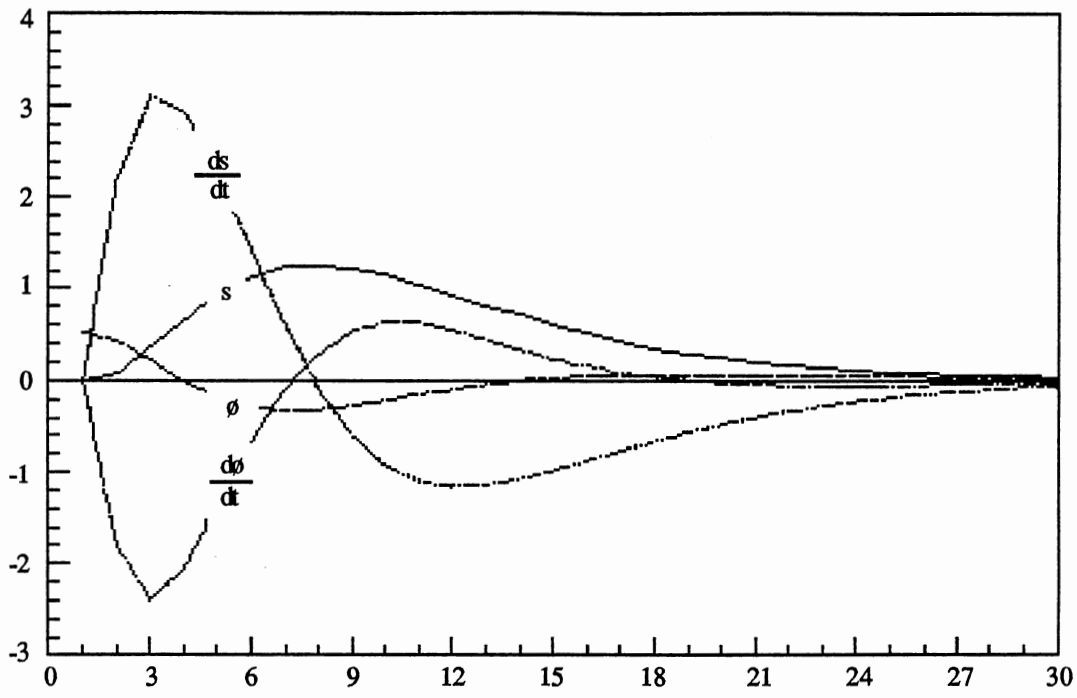


Figure 4.2. Response of System with Linear Controller

The linear controller can be seen as a sum of four linear functions f_1, f_2, f_3, f_4

$$u = f_1(s, ds/dt, \phi, d\phi/dt) + f_2(s, ds/dt, \phi, d\phi/dt) + f_3(s, ds/dt, \phi, d\phi/dt) + f_4(s, ds/dt, \phi, d\phi/dt) \quad \text{(IV.7)}$$

where:

$$f_1(s, ds/dt, \phi, d\phi/dt) = k_1 s$$

$$f_2(s, ds/dt, \phi, d\phi/dt) = k_2 \, ds/dt$$

$$f_3(s, ds/dt, \phi, d\phi/dt) = k_3 \, \phi$$

$$f_4(s, ds/dt, \phi, d\phi/dt) = k_4 \, d\phi/dt$$

Therefore, each function f_i is a linear function of one of the state variables with slope k_i.

The backpropagation algorithm was used to train a three layer neural network to reproduce the linear control law [11]. The

network used is shown in Figure 4.3. It has four inputs (the four state variables), 16 neurons in the input layer, four neurons in the hidden layer and one output neuron (control input). All neurons have the sigmoid nonlinearity

$$f(x)=\frac{1}{1+e^{-x}}$$

The output of the network is scaled to obtain the force applied to the cart ranging between specified limits, -F to F.

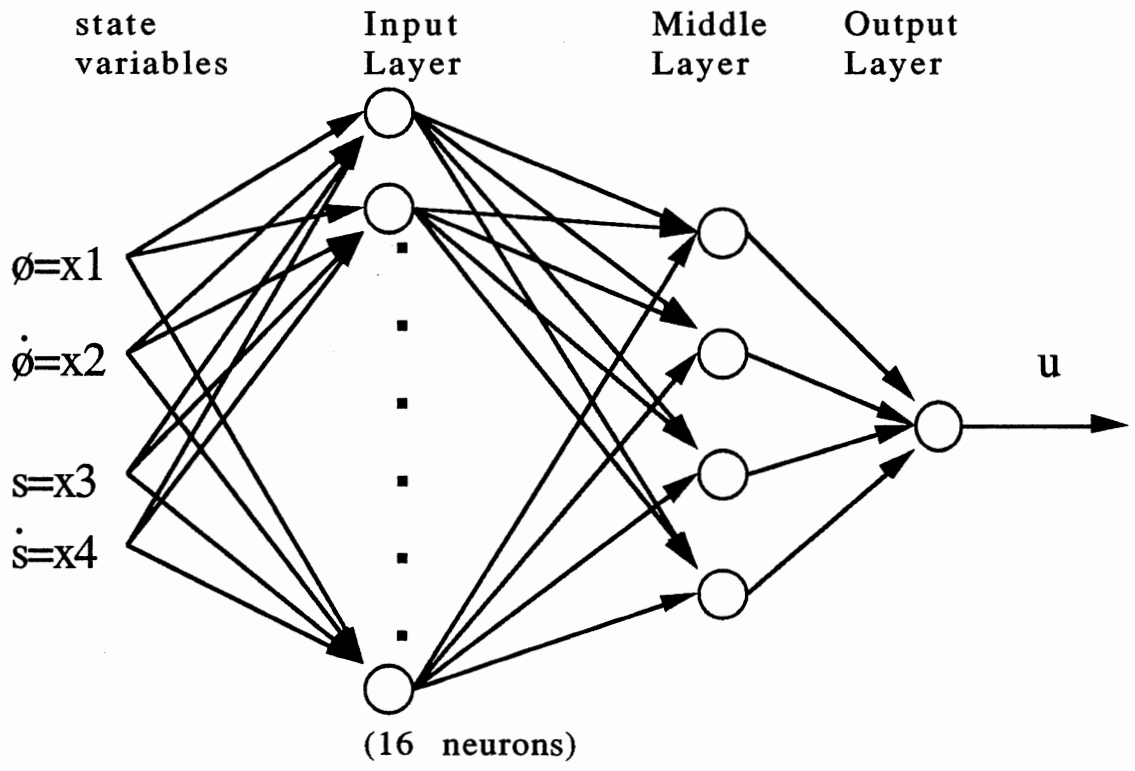


Figure 4.3. Multilayer Neural Network Controller

Figure 4.4 illustrates how this network was trained to reproduce the linear controller. Inputs were random in a range compatible to those of the operation of the system. The neural network is attempting to approximate the mapping performed by the linear controller. It is to match the functions f_1, f_2, f_3, f_4 as closely as possible.

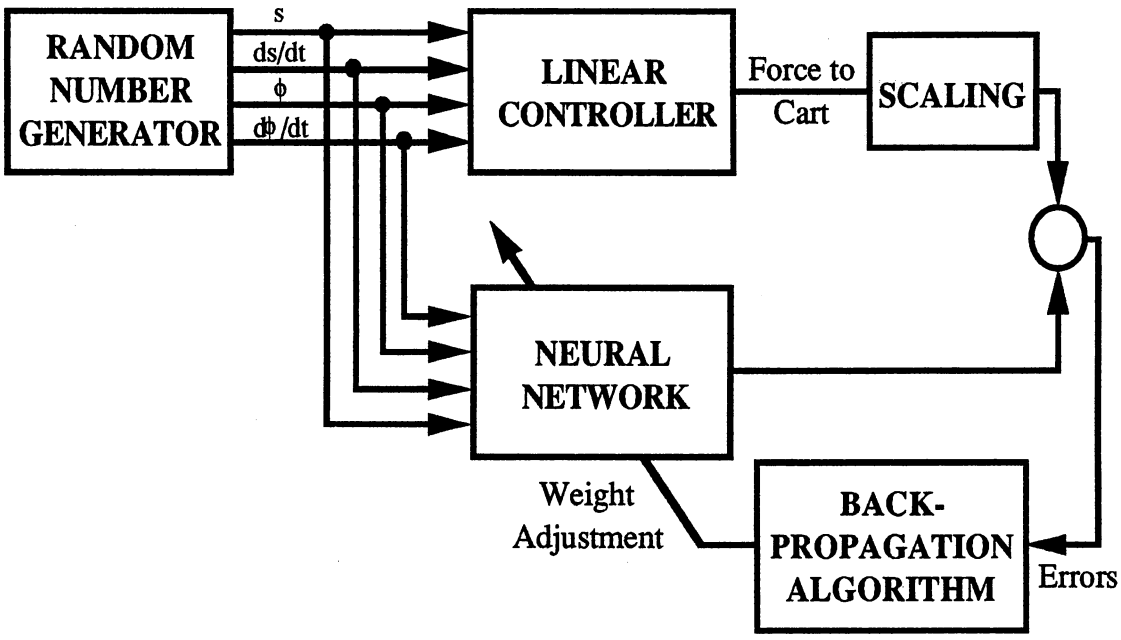


Figure 4.4. Training Mode

In order to illustrate how well the neural network can approximate the linear controller it is useful to plot the output of the network as each one of the input variables is varied (the others are set to zero), and compare that with a plot of the linear

controller. Figures 4.5 and 4.6 show the output of the linear controller (line 1) and the output of the neural network controller (line 2) as the input is varied from -4 to +4. Figure 4.5 was obtained after 300 iterations of the backpropagation algorithm were performed while Figure 4.6 was made after 600 iterations. Convergence of the algorithm is observed by comparing these two figures.

Figure 4.7 shows the response of the system when the neural network is used to implement the control law. A comparison with Figure 4.2 verifies that the performance of the neural network is similar to that of the linear controller.

A linear combiner (single neuron) would also learn a linear control law. Figure 4.8 shows a single neuron network that replaced the three layer neural network when the same training procedure was performed.

A plot of the output of the linear controller (line 1) and the output of the neural network (line 2) as each one of the state variables is varied from -4 to +4 is shown in Figure 4.9. This figure was made after 50 iterations of the backpropagation algorithm. A better performance is obtained compared to Figure 4.6. Also, convergence time was reduced because the neural network structure was simpler.

Figure 4.10 shows the response of the system when the pendulum is initially tilted by 30 degrees. This controller (single neuron) approximates better the linear controller than the three layer neural network because of its linearity. For weights of small magnitude the neural network operates in the linear region of the

sigmoid. A comparison with Figure 4.2 shows that the performance is similar to that of the linear controller.

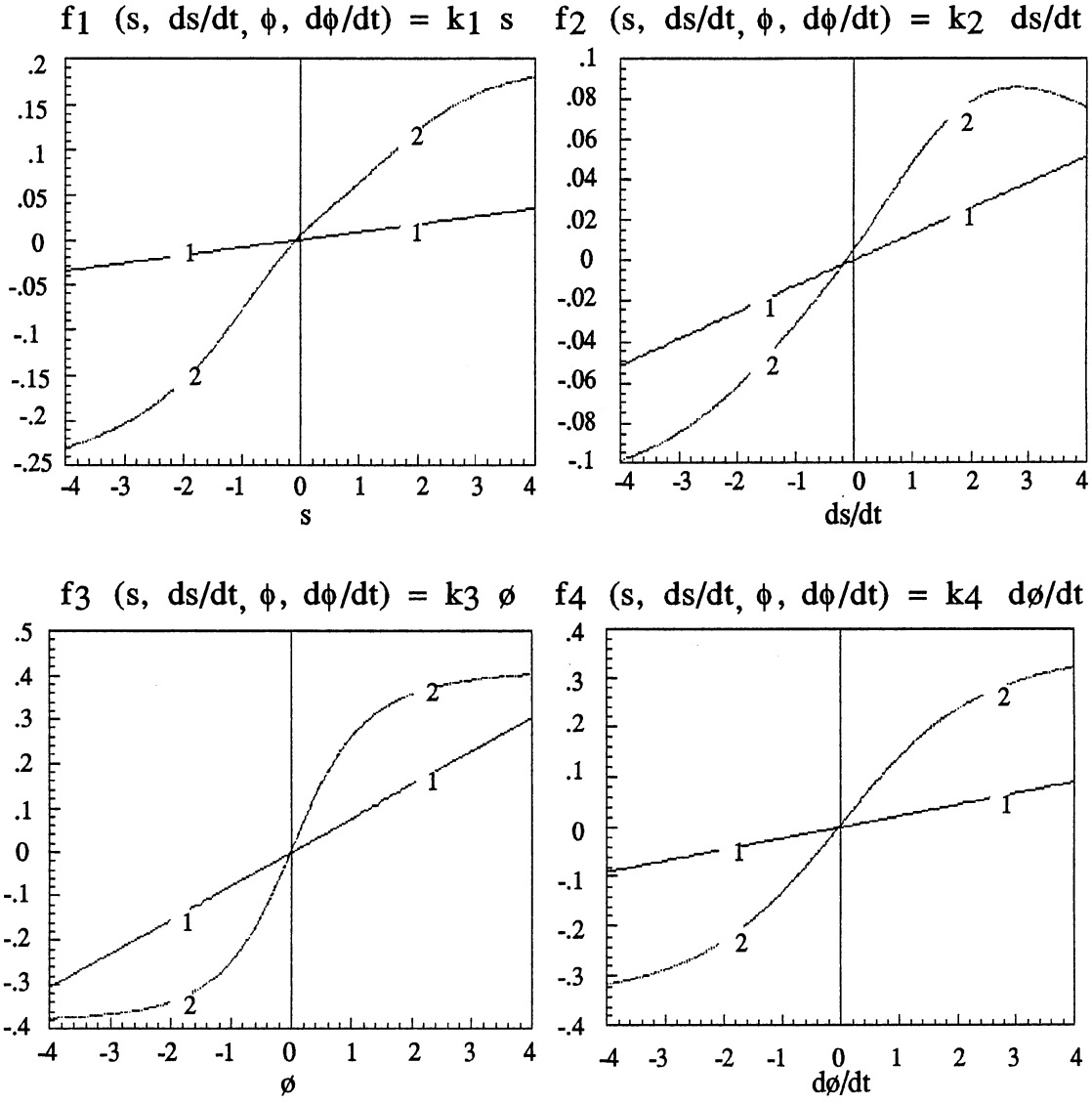


Figure 4.5. Response of the Linear Controller and the Neural Network as Each Input is Varied (After 300 Iterations)

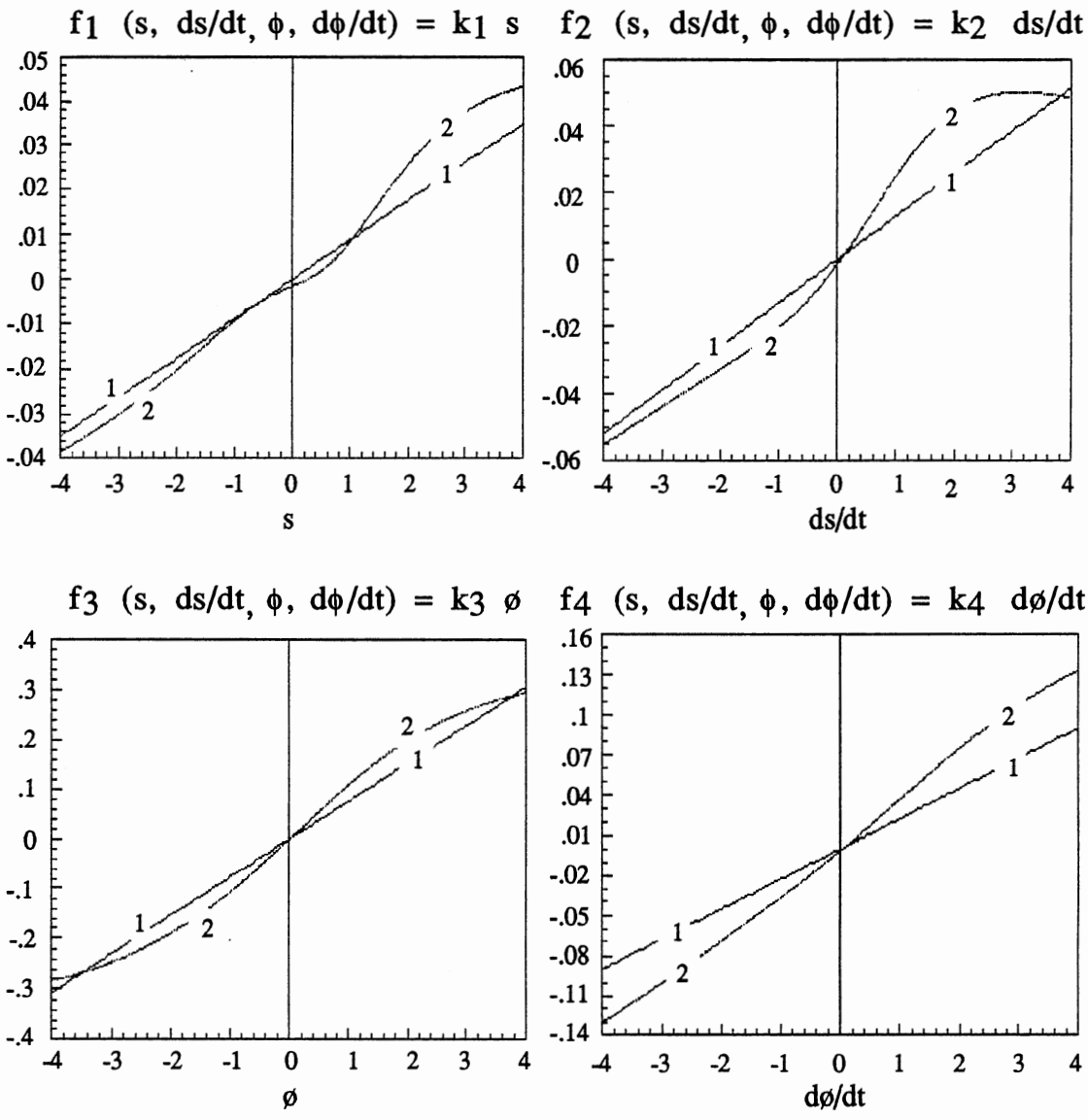


Figure 4.6. Response of the Linear Controller and the Neural Network as Each Input is Varied (After 600 Iterations)

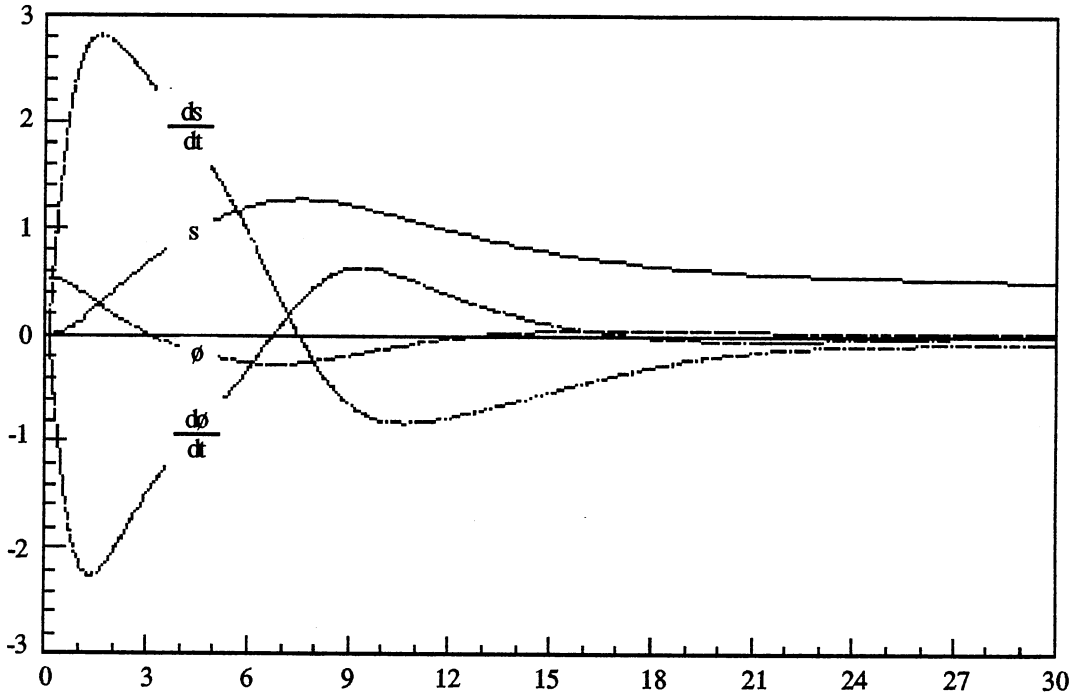


Figure 4.7. Pendulum System Response With a 3-Layer Neural Network Controller

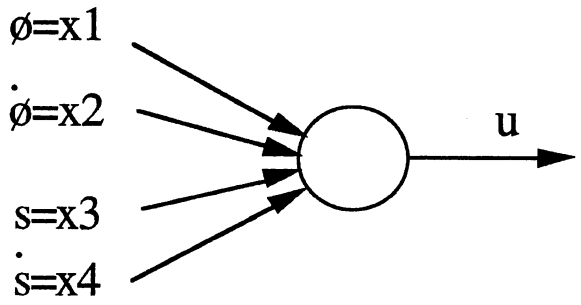


Figure 4.8. Single Neuron Controller

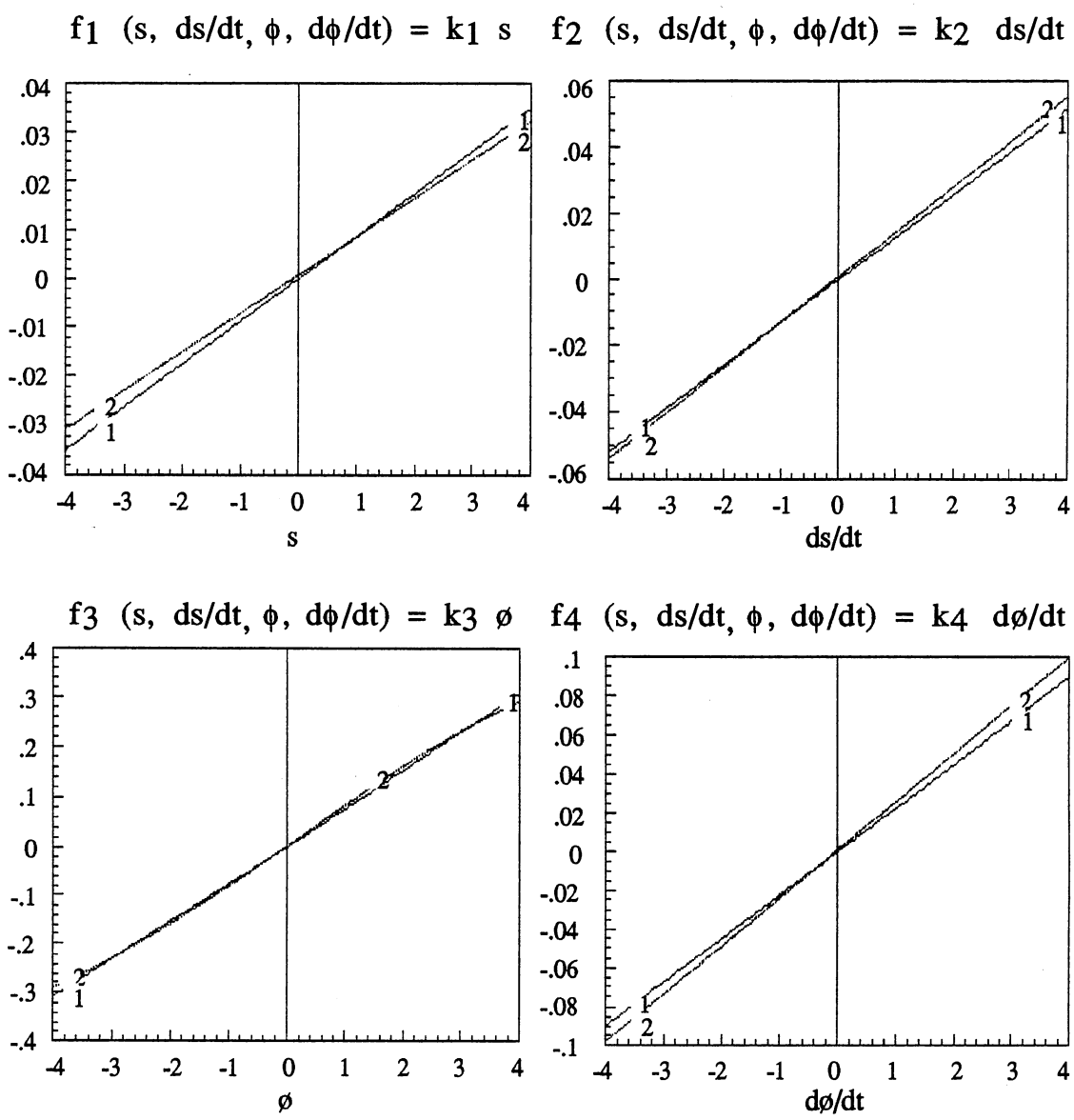


Figure 4.9. Response of the Linear Controller and the Neural Network as Each Input is Varied (After 50 Iterations)

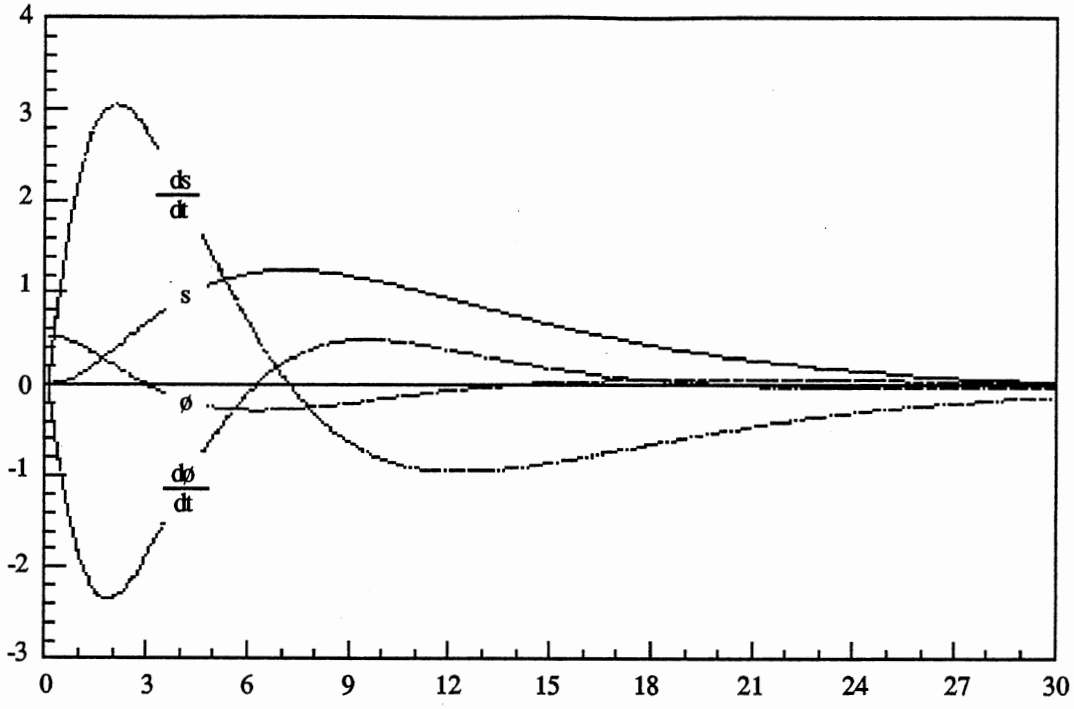


Figure 4.10. Pendulum System Response With a Single Layer Neural Network Controller

This illustrates, in a simple way, how neural networks can be implemented in control systems. The configuration of the neural network will depend largely on the function which it is approximating as was shown here and in Chapter II.

For a nonlinear plant the design of a controller is very time consuming and a substantial amount of work in the design is necessary. In the next section, neural networks are used as controllers when there is little or no knowledge of a suitable controller for the system under study. Two methods used to train these controllers are explained. Both methods have plant identification by a neural network as part of the algorithm.

Training Methods

Two training methods for control applications are going to be fully discussed in this section: Widrow's [12,13] and Narendra's [14,15]. All methods use a neural network for plant identification. Before discussing each method, plant identification is explained.

Before training the controller, a neural network is trained to imitate the plant. Since the true plant is positioned between the neural network controller and the measurable error, indirect methods of training have to be used [15]. Therefore, plant identification is done first. The plant neural network will be used to propagate back the measurable error to train the controller, backpropagation cannot be done through the true plant. Thus, in the training of the controller, the true plant and the neural network model of the plant are both used.

Figure 4.11 illustrates how the neural network model of the plant is identified. The states of the plant are assumed to be observable without noise [13]. The number of inputs of the neural network is the same as the number of states plus the number of control inputs to the plant. The number of outputs is equal to the number of states of the plant.

The neural network inputs are generated randomly with a uniform distribution. At each iteration of the learning algorithm, plant inputs u_k and states x_k are presented to the neural network and the plant. The plant gives the values of the next state x^*_{k+1} which is the desired output of the neural network. The neural

network predicts the next state x_{k+1}, and the error $(x^*_{k+1} - x_{k+1})$ is backpropagated to adjust the weights of the neural network model of the plant.

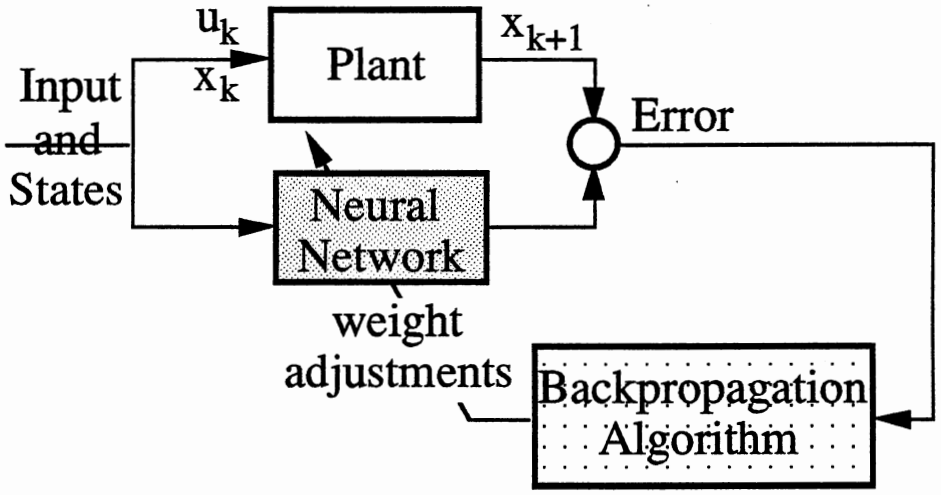


Figure 4.11. Plant Identification

When the neural network model of the plant is sufficiently accurate, the next step is to use this network in training the neural network controller. The neural network model of the plant will be used to backpropagate the measurable error so the weights of the controller can be adjusted.

Widrow's Training Method

In training the controller, the neural network model of the plant which now accurately simulates the plant dynamics, is used. The controller will be trained to give the correct input u that will drive the plant from an initial state x_o to a final desired state x_d.

The training process determines the weight values of the controller such that the performance measure J is minimized.

$$J = -\frac{1}{2}\sum_i (x_d - x_{Ni})^T (x_d - x_{Ni}) \tag{IV.8}$$

where x_{Ni} is the final state of the plant after N time steps for each initial state i. The sum is obtained over a set of initial states x_o.

Figure 4.12 shows the controller/plant box used in Figure 4.13 which illustrates the training process of the controller. The neural network controller has as many inputs as there are states and as many outputs as there are control inputs u to the true plant.

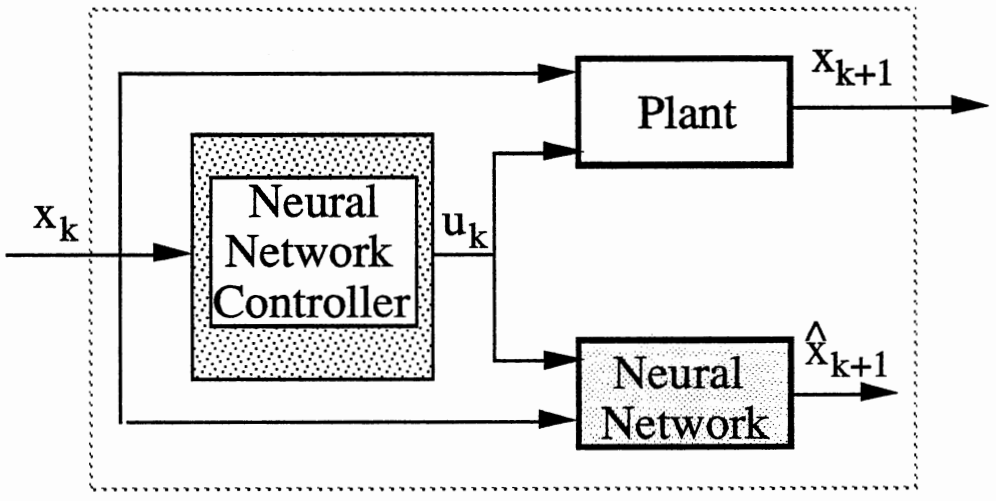


Figure 4.12. Controller/Plant Box

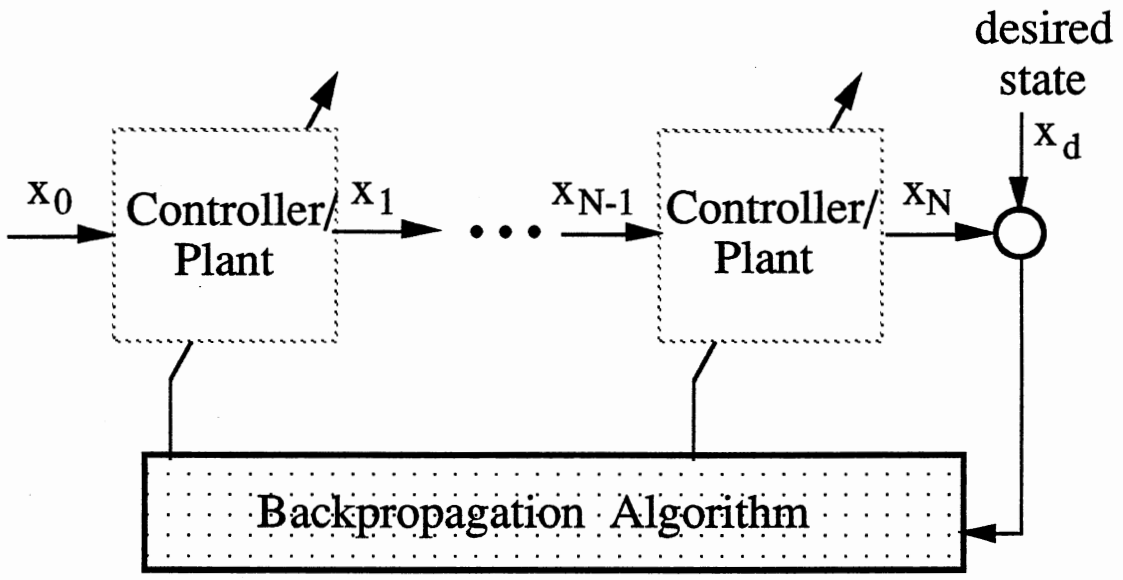


Figure 4.13. Neural Network Controller

The plant and the neural network model of the plant are used in training the controller. The plant is used to determine the output x_N after N time steps. The neural network is used to backpropagate the error $(x_d - x_N)$ to adjust the weights of the controller.

The controller, initially with random weights, gives an output u_o to the plant. The plant moves to the next state x_1. This process continues for N time steps when the plant finally reaches state x_N. The designer needs to determine the number of steps N.

To train the controller directly, the error at the output of the controller needs to be known so it can be backpropagated. Unfortunately, the desired control input to the plant is not known, so this error cannot be determined. Thus, indirect methods of training are used. The error that is known is at the output of the plant after N time steps. Since the neural network model of the

plant is also used, this error can be backpropagated through the neural network model of the plant and the controller to adjust the weights of the controller. The weights of the neural network model of the plant are fixed throughout the learning process of the controller.

The error is then backpropagated through the N steps (see Figure 4.13). The weight changes due to each step are calculated and added. The original weights of the controller are then updated.

Another approach for training the controller is discussed below.

Narendra's Training Method

The training method proposed by Narendra [14,15] begins with the procedure of plant identification, as in Widrow's method. The procedure adopted in training the controller is model reference adaptive control. In model reference control the output of the plant follows the output of a reference model. Figure 4.14 shows how this can be accomplished using a neural network model for the plant and a neural network for the controller. The neural network model of the plant and the controller receive the actual outputs of the true plant as feedback signals. The reference input is a bounded signal. The plant together with the controller should behave like the reference model.

The algorithm used to train the controller is backpropagation. The error between the reference model and the plant is backpropagated through the neural network model of the plant. It

is assumed that the order of the plant is known and the reference model has the same relative degree as that of the plant.

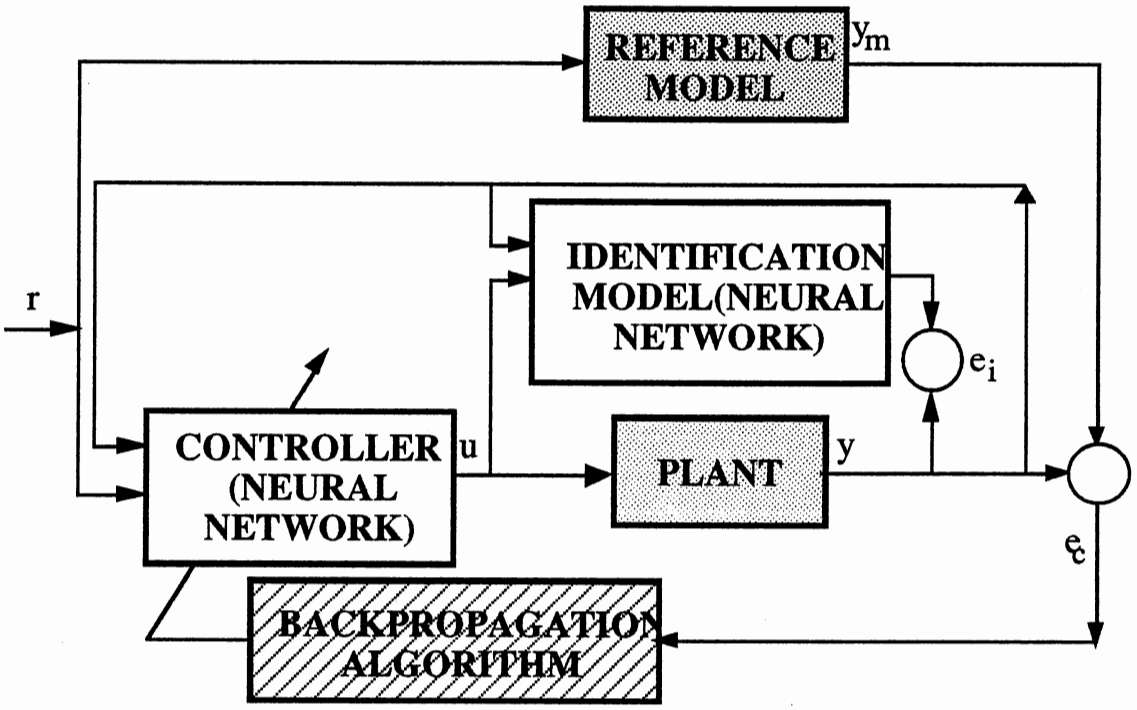


Figure 4.14. Controller Training Mode

The plant is assumed to belong to one of the following models

model (i)

$$y(k+1) = \sum a_i\, y(k-i) + g[u(k), \ldots, u(k-m)]$$

model (ii)

$$y(k+1) = f[y(k), \ldots, y(k-n)] + \sum \beta_j\, u(k-j)$$

model (iii)

$$y(k+1) = f[y(k), \ldots, y(k-n)] + g[u(k), \ldots, u(k-m)]$$

model (iv)

$$y(k+1) = f[y(k), \ldots, y(k-n), u(k), \ldots, u(k-m)]$$

where f and g are differentiable functions and $m \leq n$. The plant and reference model are assumed to be BIBO stable.

Depending on the available knowledge of the plant, different steps in the training procedure are taken. For example, a plant described by model (iii) was identified successfully by two neural networks, one for f(.) and one for g(.), N_f and N_g respectively. So,

$$\begin{aligned} y(k+1) &= f[y(k), \ldots, y(k-n)] + g[u(k)] \\ &= N_f[y(k), \ldots, y(k-n)] + N_g[u(k)] \end{aligned} \qquad \text{(IV.9)}$$

The reference model is

$$y_m(k+1) = \Sigma_i \, ß(i) \, y_m(i) + r(k) \qquad \text{(IV.10)}$$

In model reference control the output of the plant is equal to the output of the reference model, therefore

$$y(k+1) = y_m(k+1) \qquad \text{(IV.11)}$$

Substituting Equations IV.9 and IV.10 in Equation IV.11 gives

$$N_f[y(k), \ldots, y(k-n)] + N_g[u(k)] = \Sigma_i \, ß(i) \, y_m(i) + r(k)$$

Rearranging

$$N_g[u(k)] = -N_f[y(k), \ldots, y(k-n)] + \Sigma_i \, ß(i) \, y_m(i) + r(k) \qquad \text{(IV.12)}$$

If N_c is the neural network obtained such that $N_c[N_g(u)] \approx u$, then N_g composed with N_c is an identity mapping. Applying N_c to both sides of Equation IV.12 one obtains

$$N_c[N_g[u(k)]] = N_c[-N_f[y(k), \ldots, y(k-n)] + \Sigma_i \, ß(i) \, y_m(i) + r(k)]$$

Therefore,

$$u(k) = N_c[-N_f[y(k), \ldots, y(k-n)] + y_m(k+1)]$$

Figure 4.15 illustrates how this controller is implemented where z^{-1} represent delays. For a numerical example of this procedure, see [14].

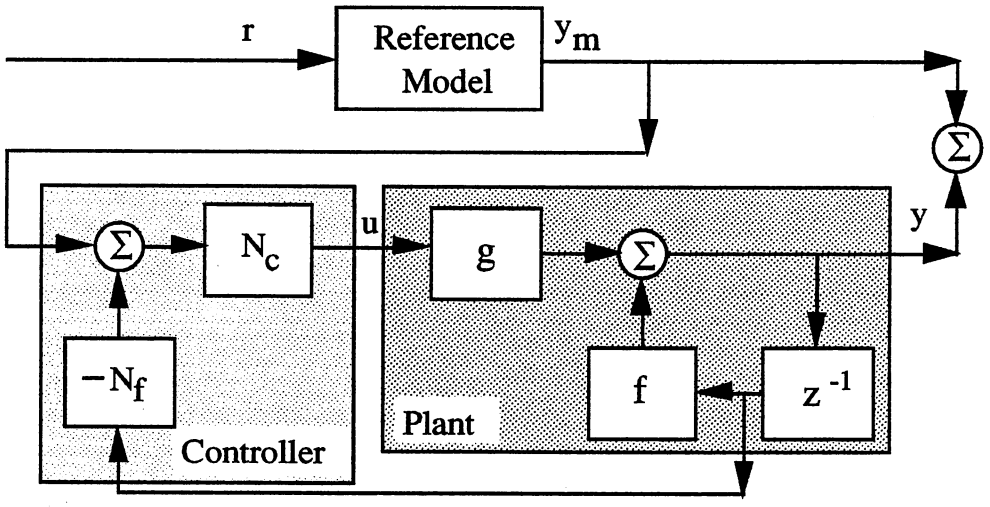


Figure 4.15. Neural Network Controller with Nonlinear Plant

Widrow's method assumes that there is no knowledge of the plant, therefore in Narendra's classification the plant is of class (iv) (a general nonlinear system). In the next Chapters these two methods will be used to determine a controller for a simplified model of the Extended Range Gun.

CHAPTER V

WIDROW'S METHOD APPLIED TO THE EXTENDED RANGE GUN

The Extended Range Gun (ERG) is a flexible weapon with a large unbalance. Figure 5.1 shows a schematic of the ERG. The control of this weapon is made difficult because of its flexibility and its distributed nature. Its vibrational modes are numerous, densely packed, and relatively low frequency. The control problem is further complicated by the need to design a controller which has low enough order so that it can be implemented on an onboard computer and yet of high enough order to provide accurate control.

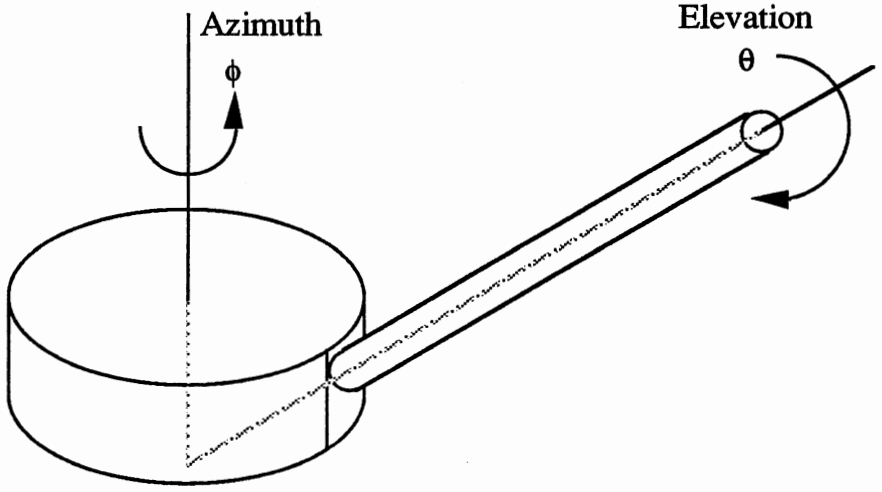


Figure 5.1. Schematic of the Extended Range Gun

The simplest model of the ERG is shown in Figure 5.2. It represents the ERG as a completely rigid system. This model will be used in this chapter and the following chapter to illustrate the control design process for neural network controllers. In the next stage of this research more complex models will be used. In these more complex models, the elastic modes of the ERG, will be included.

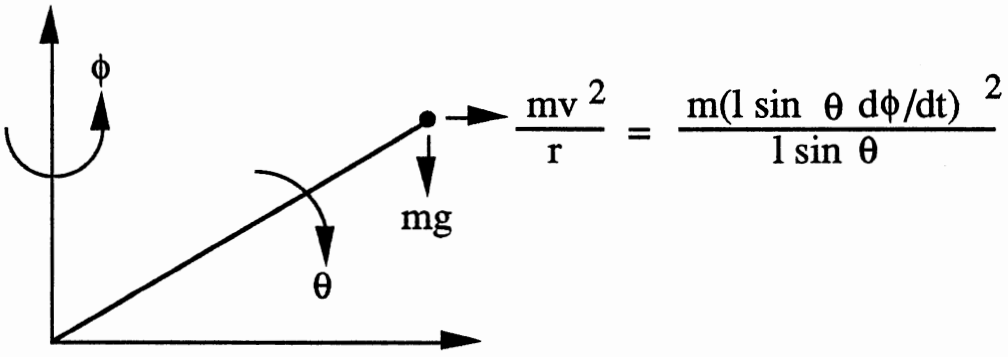


Figure 5.2. Simplified Model of the Extended Range Gun

This chapter describes the design of a nonlinear controller to provide the motor which drives the gun with the correct input current. It is desired that the controller drive the gun from any initial position to a desired final position. In this research a neural network controller is used. This neural network is trained using the algorithms discussed in Chapter IV. Widrow's and Narendra's

methods are compared as the neural network is trained to control the simplified model of the ERG. Widrow's method will be presented in this chapter, and Narendra's method will be studied in the next chapter.

It will be assumed that the equations that govern the movement of the ERG are the following

$$\frac{d}{dt}\left(J_a(\theta)\frac{d\phi}{dt}\right) = -\beta_a \frac{d\phi}{dt} + k_a u_a$$

$$J_e \frac{d^2\theta}{dt^2} = -\beta_e \frac{d\theta}{dt} + k_e u_e + mgl * \sin(\theta) + ml^2 \sin(\theta)\cos(\theta)\left(\frac{d\phi}{dt}\right)^2 \quad \text{(V.1)}$$

where: θ - elevation angle (radians)

ϕ - azimuth angle (radians)

m - mass of the gun concentrated at the endpoint (=2 kg)

l - length of gun (=1 m)

β_a and β_e - viscous friction coefficient (=4 Nms)

k_a and k_e - motor torque constant (=2 Nm/A)

g - acceleration due to gravity (=10 m/s^2)

J_a - moment of inertia about the azimuth (=ml^2 sin^2(θ))

J_e - moment of inertia about the elevation (=ml^2)

u_a - input current to the motor that moves the gun in θ

u_e - input current to the motor that moves the gun in ϕ

It is assumed that the ERG is allowed to rotate completely around the ϕ axis but θ is only allowed to go from $0+\mu$ to $\pi-\mu$, where μ is a small number. The elevation limits are larger than would be practical, but they do not significantly affect the results which are

reported in this chapter. Also, the input current to the motors that drive the gun are assumed to saturate at 20A.

Substituting the values of the constants into Equation V.1, the state variable form of the model can be written as follows. Let the state variables be: $x_1 = \theta$, $x_2 = d\theta/dt$, $x_3 = \phi$, $x_4 = d\phi/dt$ then

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 \sin(x_1) - 2 x_2 + x_4^2 \sin(x_1) \cos(x_1) + u_e$$

$$dx_3/dt = x_4$$

$$dx_4/dt = (-2 x_2 x_4 \sin(x_1) \cos(x_1) - 2 x_2 + u_a)/\sin^2(x_1) \qquad (V.2)$$

This is the complete reduced model of the ERG. In the simulations that follow either the elevation angle or the azimuth angle is held constant. When the elevation angle is constant, the azimuth model is obtained. The elevation model is derived when the azimuth angle is constant.

Azimuth Model

The azimuth model (θ fixed) is shown in Figure 5.3. It is a second order linear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$

$$dx_2/dt = (- 2 x_2 + u)/\sin^2(\theta) \qquad (V.3)$$

where: $x_1 = \phi$

$x_2 = d\phi/dt$

u - input current to the motor that moves the gun in ϕ

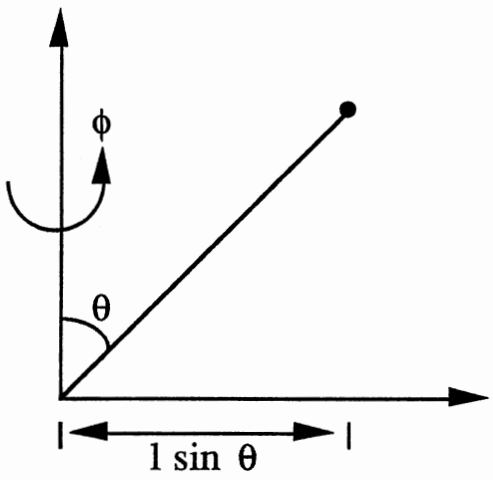


Figure 5.3. Azimuth Model

In this study θ is fixed at $\pi/2$ (90°) and Euler's integration method is used to obtain x_1 and x_2 where $x(k+1) = x(k) + \Delta t * dx/dt$ with $\Delta t = 0.01$ sec, the sampling interval.

As described in Chapter IV both methods used to train the controller have two stages. The first stage is plant identification and the second stage is the actual training of the controller. Figure 4.11 shows a sketch of the procedure for plant identification.

The neural network plant has a single layer, with 3 inputs which correspond to the 2 state variables and the input current, and 2 outputs which correspond to the 2 state variables as shown in Figure 5.4. Figure 5.4 also shows the architecture of the controller. It has a single layer with one neuron. The 2 inputs correspond to the 2 state variables, and the output corresponds to the input current. All neurons have a linear transfer function,

f(x) = x. The saturation of the controller's output at 20 A is handled by a separate network.

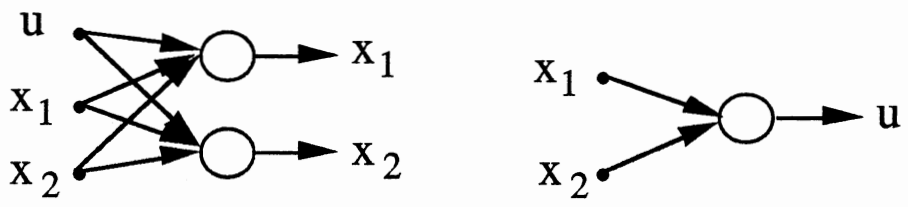


Figure 5.4. Plant and Controller Architecture

In training the plant, inputs and states are generated randomly. These are presented to the neural network plant and true plant to obtain the predicted and desired next states. An error is calculated and backpropagated allowing the weights of the neural network to be adjusted. This is done for many iterations until a performance criterion is satisfied. The learning curve for the plant is shown in Figure 5.5. The plant neural network is then used to train the controller.

The output of the controller is saturated. Therefore, the saturation function must be first learned by a neural network before training the controller. This is necessary so that the output error can be backpropagated and the weights of the controller changed. The saturation network has two layers, with four neurons in the hidden layer. There is only one input and one output.

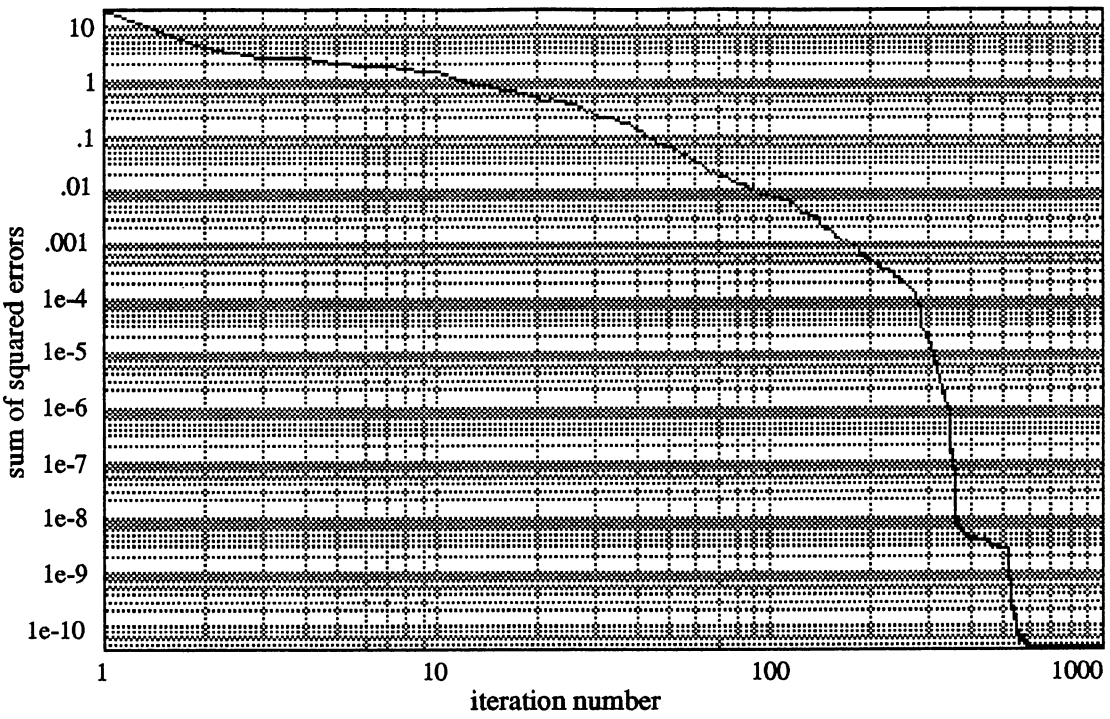


Figure 5.5. Learning Curve for Plant ($\theta = \pi/2$)

Figure 5.6 shows the learning curve for the saturation network and Figure 5.7 shows the transfer function for the saturation function, f(x), and saturation network, N[x]. As one can see the neural network learned to reproduce the saturation function. The saturation network is then used to obtain the controller.

Figure 4.13 schematically shows the training of the controller when Widrow's method is used. The initial state vector x_o is generated randomly. The controller calculates the input current to the motor. The saturation network saturates this current and the plant moves to the next state. This cycle proceeds for N steps (N is fixed at 200, chosen arbitrarily). The final state x_N is then

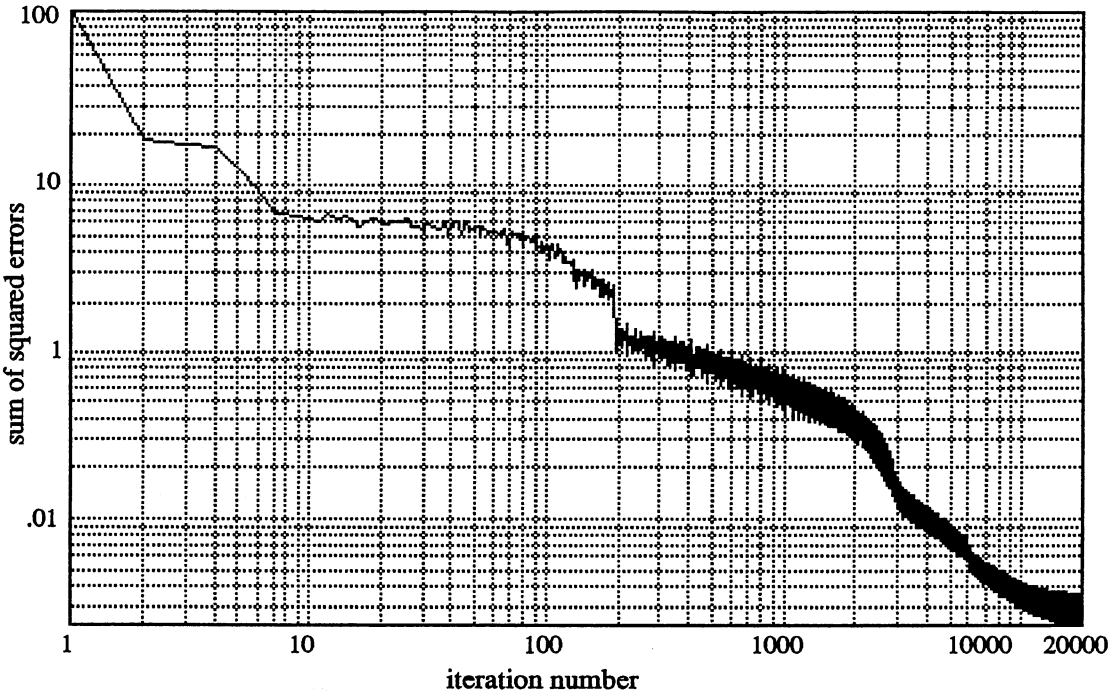


Figure 5.6. Learning Curve for the Saturation Network

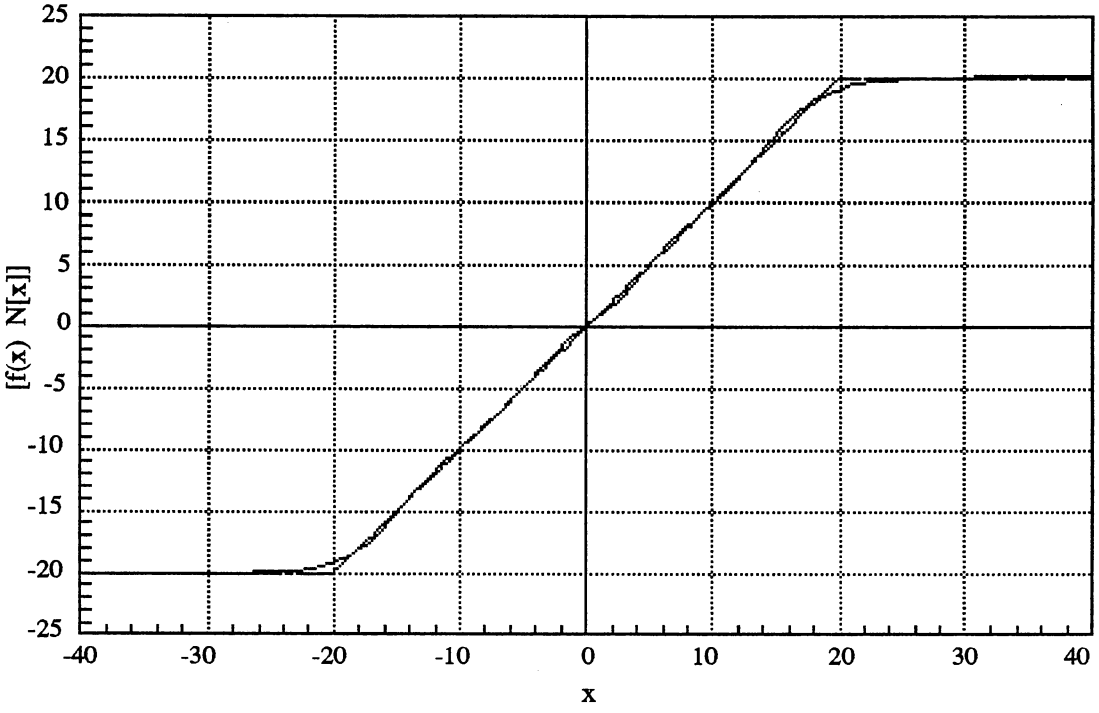


Figure 5.7 Saturation Transfer Function

compared with the desired state $x_d = [0 \ \ 0]^T$ to obtain the error which is backpropagated to adjust the weights of the controller. The error is backpropagated through the neural network plant but the true plant is used to determine the output error itself. This process is repeated by choosing another initial condition.

The weights of the controller are initially random. They remain constant throughout the N steps. The weights are allowed to change by the backpropagation algorithm only after the N steps are performed. The learning curve for the controller is shown in Figure 5.8. The increase in the sum of squared errors at 50 iterations is due to an increase in the range of the initial conditions of the plant.

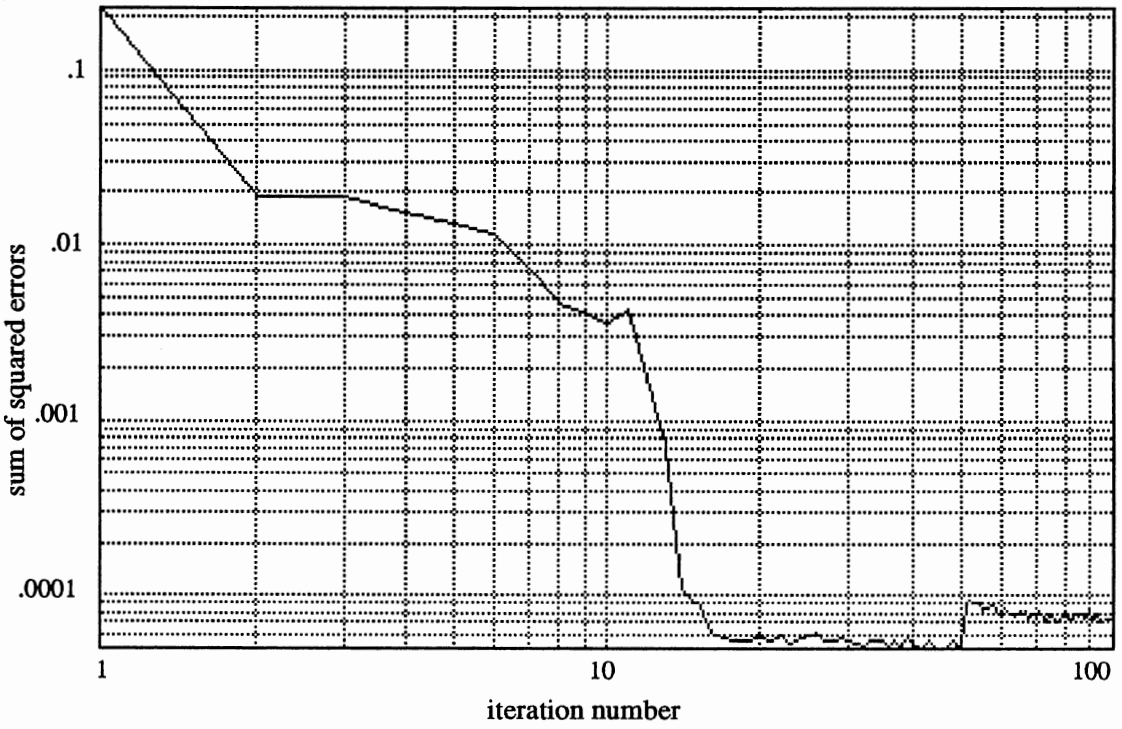


Figure 5.8. Learning Curve for the Controller

This controller was tested to control the true plant (azimuth angle only). Figure 5.9 shows the operational mode of the system, where z^{-1} represents a delay, with the saturation and neural network controller. The response of the system for an initial condition of $[3 \quad 0]^T$ is given in Figure 5.10 where x_1 is the azimuth angle, $x_2 = dx_1/dt$ is velocity, dx_2/dt is acceleration and u is the controlled input. As one can see all states go to zero as desired.

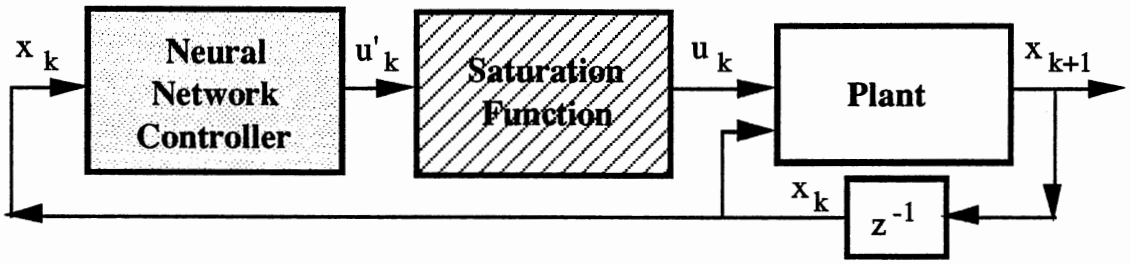


Figure 5.9. Operational Mode of the System

Elevation Model

The elevation model (ϕ fixed) is shown in Figure 5.11. It is a second order nonlinear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 \sin(x_1) - 2 x_2 + u \qquad \text{(V.4)}$$

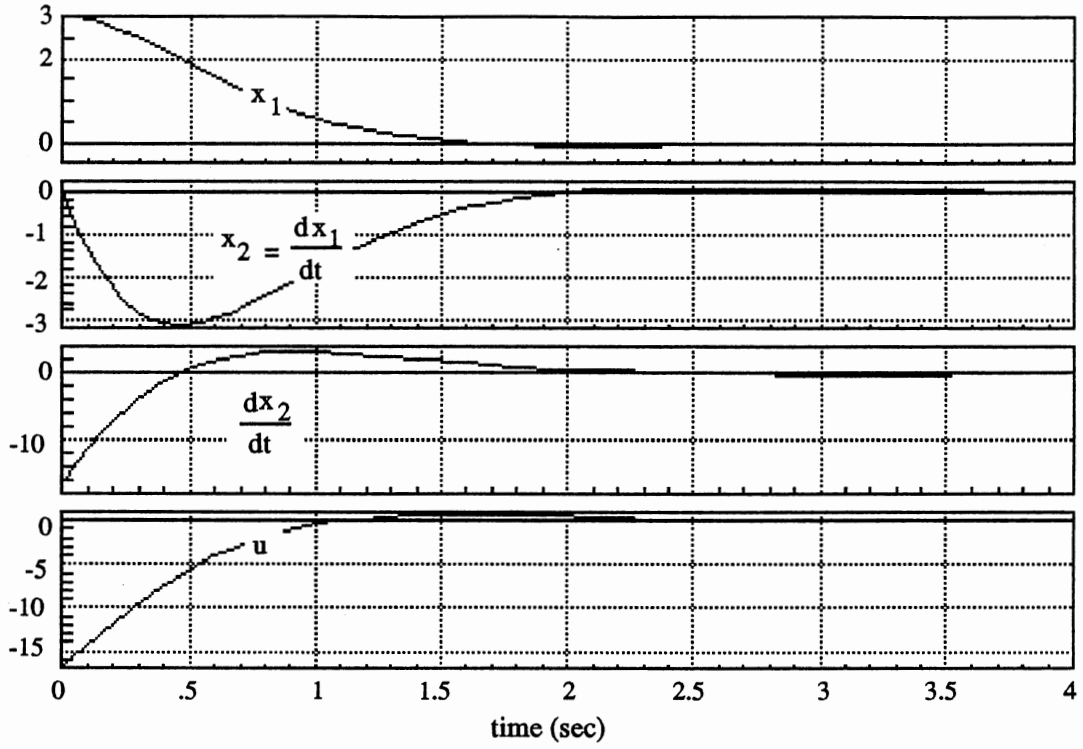


Figure 5.10. Response for Initial Condition $[3 \quad 0]^T$

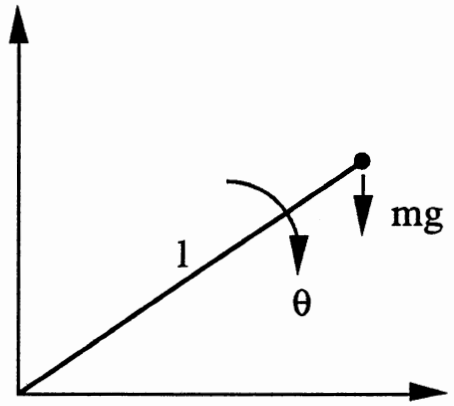


Figure 5.11. Elevation Model

where: $x_1 = \theta$

$x_2 = d\theta/dt$

u - input current to the motor that moves the gun in θ

The integration method used to obtain x_1 and x_2 was Euler's method where $x(k+1) = x(k) + \Delta t * dx/dt$ with $\Delta t = 0.01$ sec.

Two algorithms were used when Widrow's method was employed: modified backpropagation and conjugate gradient. These two algorithms are compared in the following section.

The Backpropagation Algorithm

The backpropagation algorithm was applied first to train the plant and controller. The neural network plant has two layers, with 24 neurons in the hidden layer. There are 3 inputs corresponding to the 2 state variables and the input current, and 2 outputs corresponding to the 2 state variables as shown in Figure 5.12. Figure 5.12 also shows the architecture of the controller. It has two layers with 24 neurons in the hidden layer. The 2 inputs correspond to the 2 state variables and the output corresponds to the input current. All neurons have a sigmoid transfer function.

In training the plant, inputs and states are generated randomly. These are presented to the neural network plant and true plant to obtain the predicted and desired next states. The output of the plant needs to be scaled in order to obtain the error. This error is backpropagated allowing the weights of the neural network to be adjusted. This is done for many iterations until a performance criterion is satisfied. The learning curve for the plant is shown in

Figure 5.13. The plant neural network is then used to train the controller.

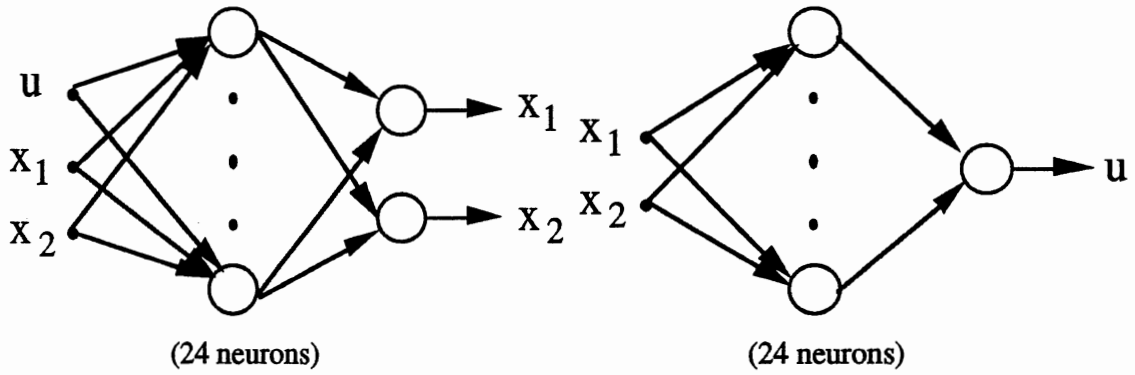


Figure 5.12. Plant and Controller Architecture

Figure 4.13 schematically shows the training of the controller when Widrow's method is used. The initial state vector x_o is generated randomly. The controller calculates the input current to the motor. The current is scaled and input to the plant. The plant moves to the next state. This cycle proceeds for N steps where N depends on the initial state x_o with a maximum value of 120. The final state x_N is then compared with the desired state $x_d = [\pi/2 \quad 0]^T$ to obtain the error which is backpropagated to adjust the weights of the controller. The error is backpropagated through the neural network plant but the true plant is used to determine the output error itself. This process is repeated by choosing another initial condition.

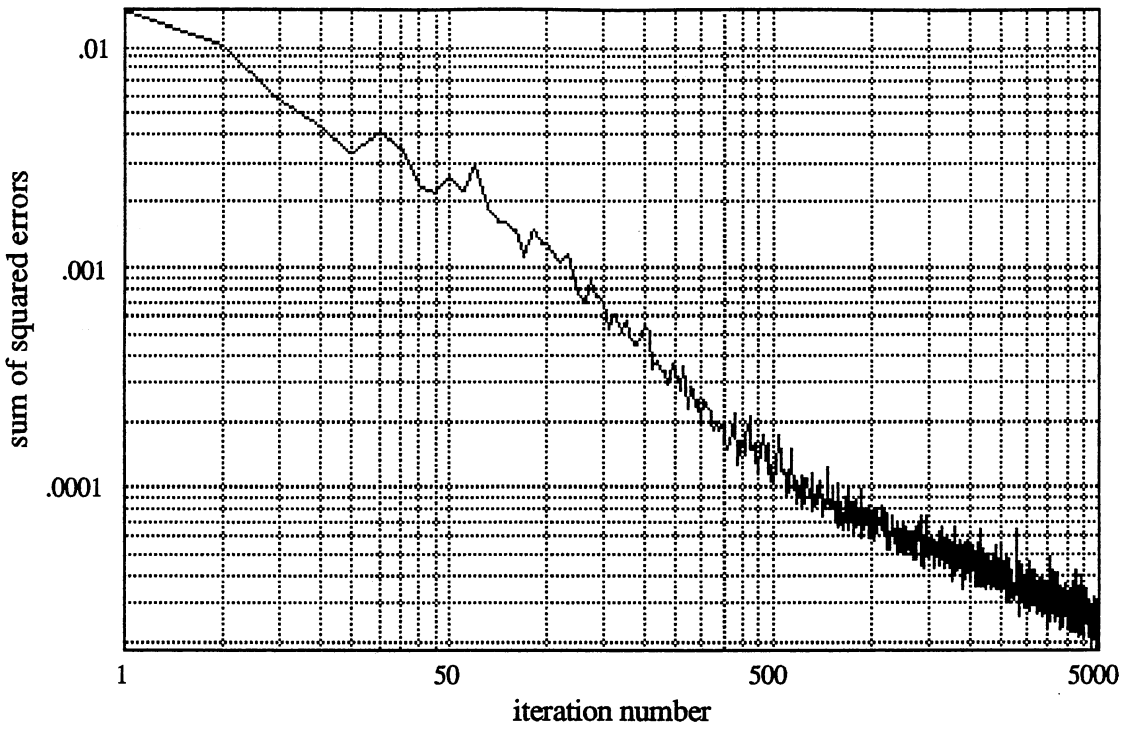


Figure 5.13. Learning Curve for Plant

The weights of the controller are initially random. They remain constant throughout the N steps. The weights are allowed to change by the backpropagation algorithm only after the N steps are completed.

The elevation angle θ can only have values ranging from $0+\mu$ to $\pi-\mu$ where μ is a small number. If the elevation angle goes beyond these limits the system stops. The learning of the controller is affected by the criterion chosen to determine x_N when the gun hits a limit. If the gun hits a limit during training, three different final conditions can occur as shown in Figure 5.14.

Position I occurs when the final position of the gun is taken at the stops. In this case x_N = limit and N = L where L is the number of steps taken when the limit is passed. Position II occurs when the final position of the gun is taken before the stops, which gives x_N = x_{L-1} and N = L - 1. Finally, position III occurs when the final position of the gun is taken after the stops, which gives $x_N = x_L$ and N = L. Figure 5.15 shows the learning curve for the controller when case I is considered. The learning curve for case II is shown in Figure 5.16. Finally, case III is shown in Figure 5.17.

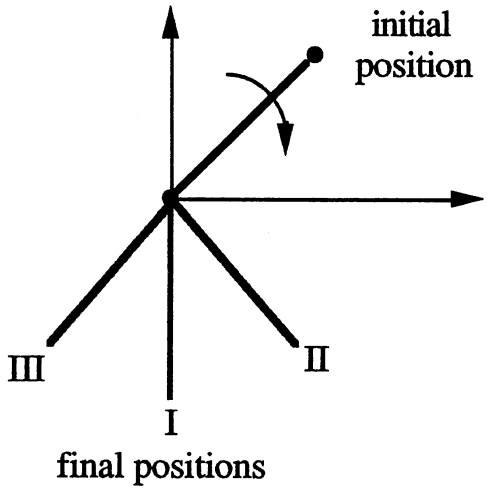


Figure 5.14. Final Positions of the Gun

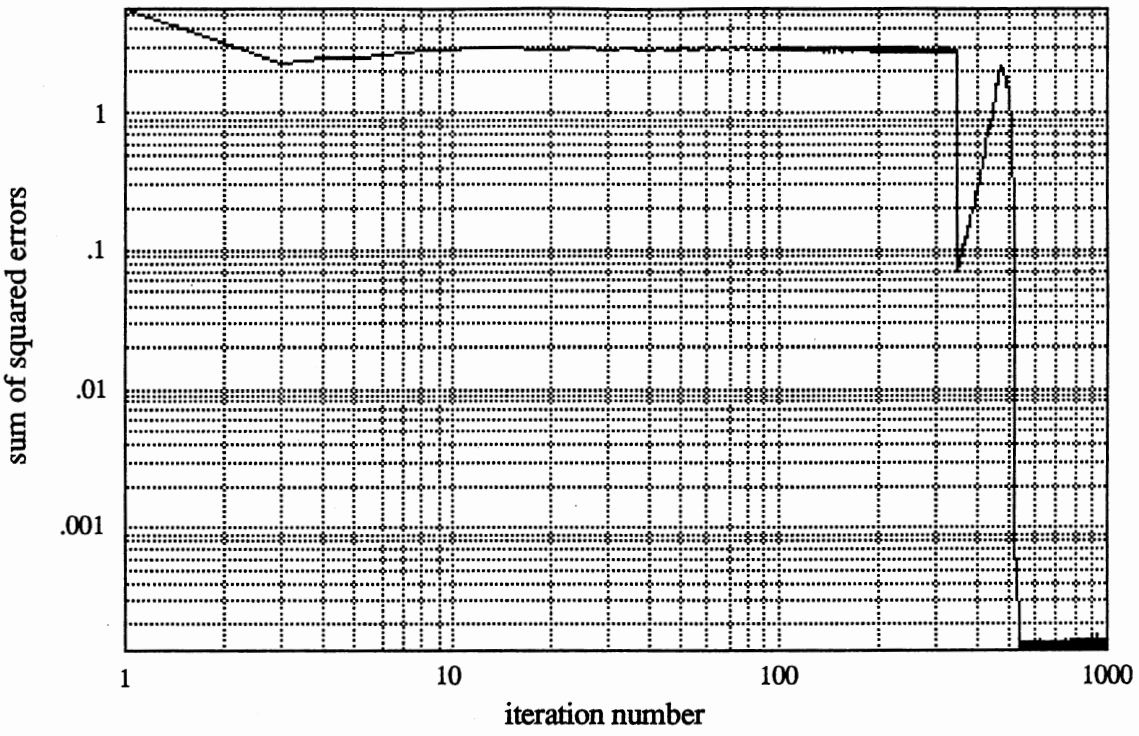


Figure 5.15. Learning Curve for Controller (case I)

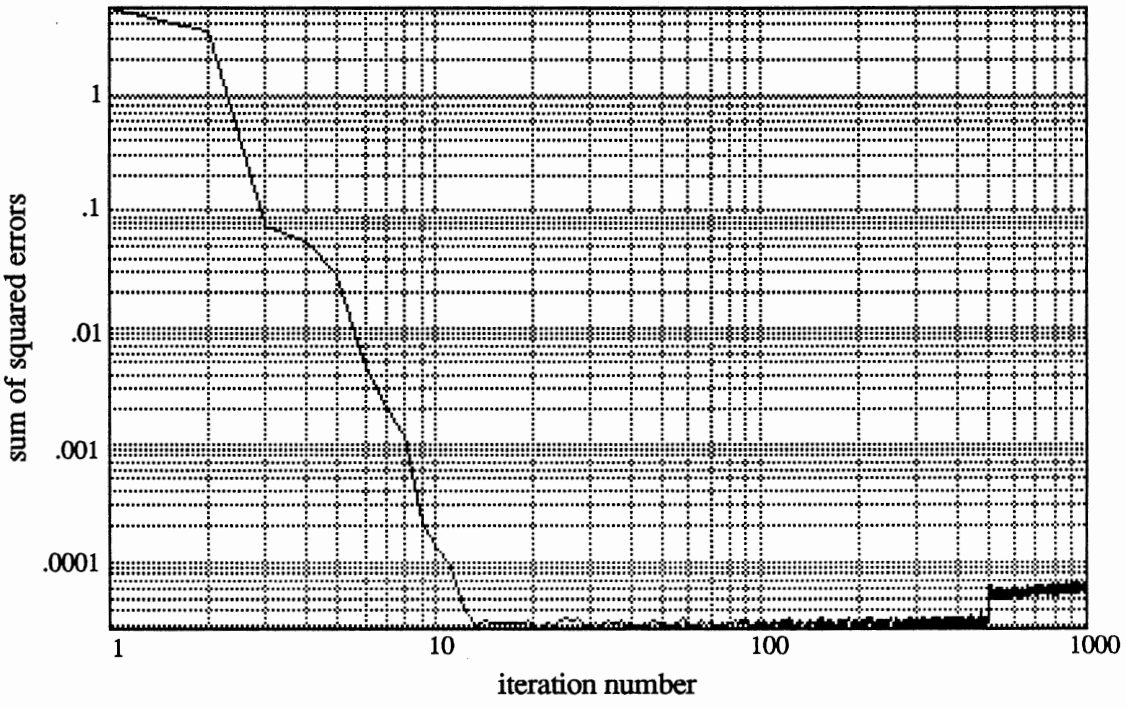


Figure 5.16. Learning Curve for Controller (case II)

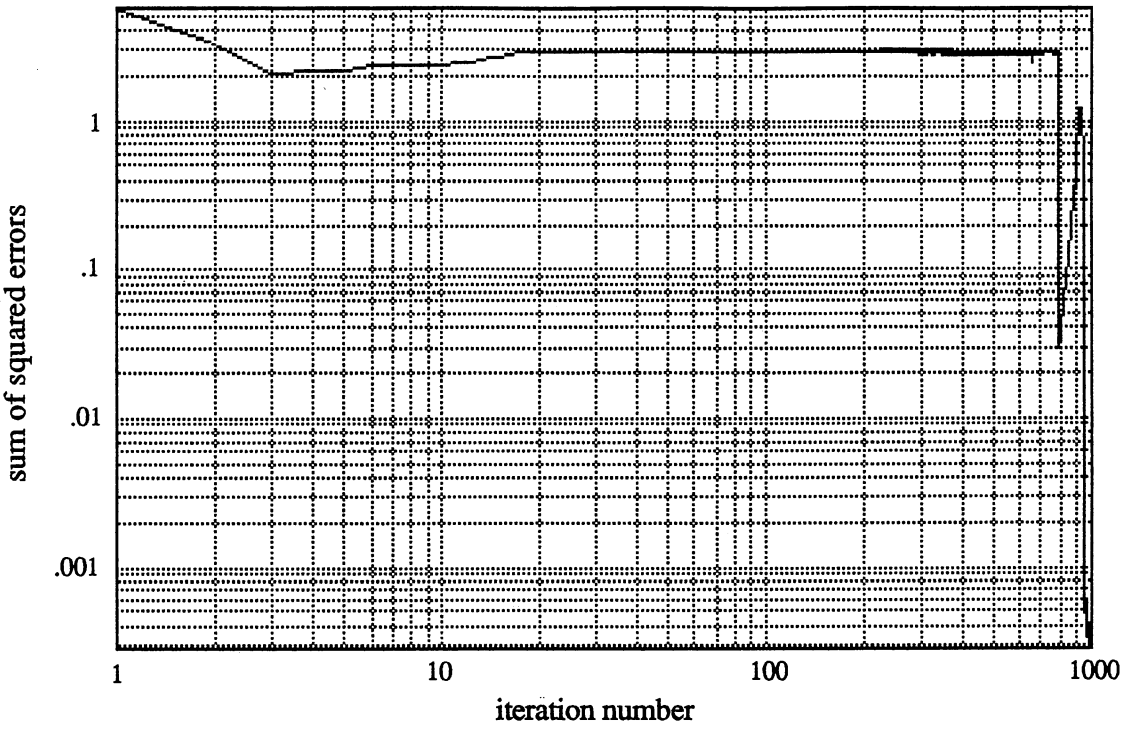


Figure 5.17. Learning Curve for Controller (case III)

The Conjugate Gradient Algorithm

The elevation model was also trained using the conjugate gradient algorithm with line search. The equations in state variable form that describe the system are again shown below.

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 \sin(x_1) - 2 x_2 + u \qquad \text{(V.5)}$$

where: $x_1 = \theta$

$x_2 = d\theta/dt$

u - input current to the motor that moves the gun in θ

The integration method used to obtain x_1 and x_2 was Euler's method where $x(k+1) = x(k) + \Delta t * dx/dt$ with $\Delta t = 0.01$ sec.

The neural network plant and neural network controller have two layers, with 24 neurons in the hidden layer as shown in Figure 5.12 above. The three inputs of the plant correspond to the 2 state variables and the input current, and the 2 outputs correspond to the 2 state variables. The two inputs to the controller are the 2 state variables and the output corresponds to the input current. The hidden neurons have a sigmoid type transfer function and the output neurons have a linear transfer function.

In training the plant, inputs and states are generated randomly. These are presented to the neural network plant and true plant to obtain the predicted and desired next states. Since the output layer is linear, no scaling is necessary. An error is calculated and backpropagated allowing the weights of the neural network to be adjusted. This is done for many iterations until a performance criterion is satisfied. The learning curve for the plant is shown in Figure 5.18. The plant neural network is then used to train the controller.

In training the controller, the initial state x_o is generated randomly. The controller calculates the input current to the motor. This current is saturated and input to the plant. The plant moves to the next state. This cycle proceeds for N steps where N depends on the initial state x_o with a maximum value of 120. The final state x_N is then compared with the desired state $x_d = [\pi/2 \quad 0]^T$ to obtain the error which is backpropagated to adjust the weights of the controller. The error is backpropagated through the neural

network plant and saturation but the true plant is used to determine the output error itself. This process is repeated by choosing another initial condition.

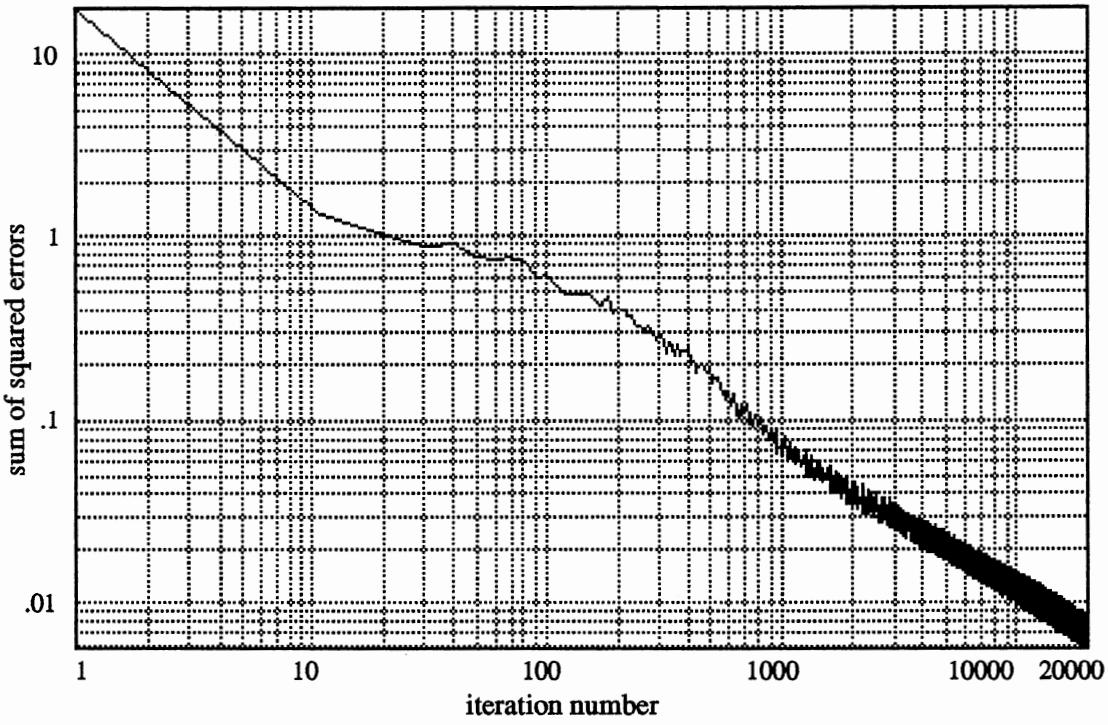


Figure 5.18. Learning Curve for the Plant

To backpropagate the error to the controller the saturation function needs to be identified by a neural network. The learning of the saturation function is shown in Figure 5.6. Figure 5.7 illustrates the transfer function for the saturation function and the saturation neural network.

As explained earlier, the elevation angle θ can only have values ranging from $0+\mu$ to $\pi-\mu$. If the elevation angle goes beyond these limits the system stops. This stopping criterion allows the final position x_N of the gun to be selected in different ways. Figure 5.14 shows the three different final conditions that the gun can take when a limit is hit. When the conjugate gradient algorithm is used this criterion does not affect the learning of the controller as illustrated in Figures 5.19 and 5.20.

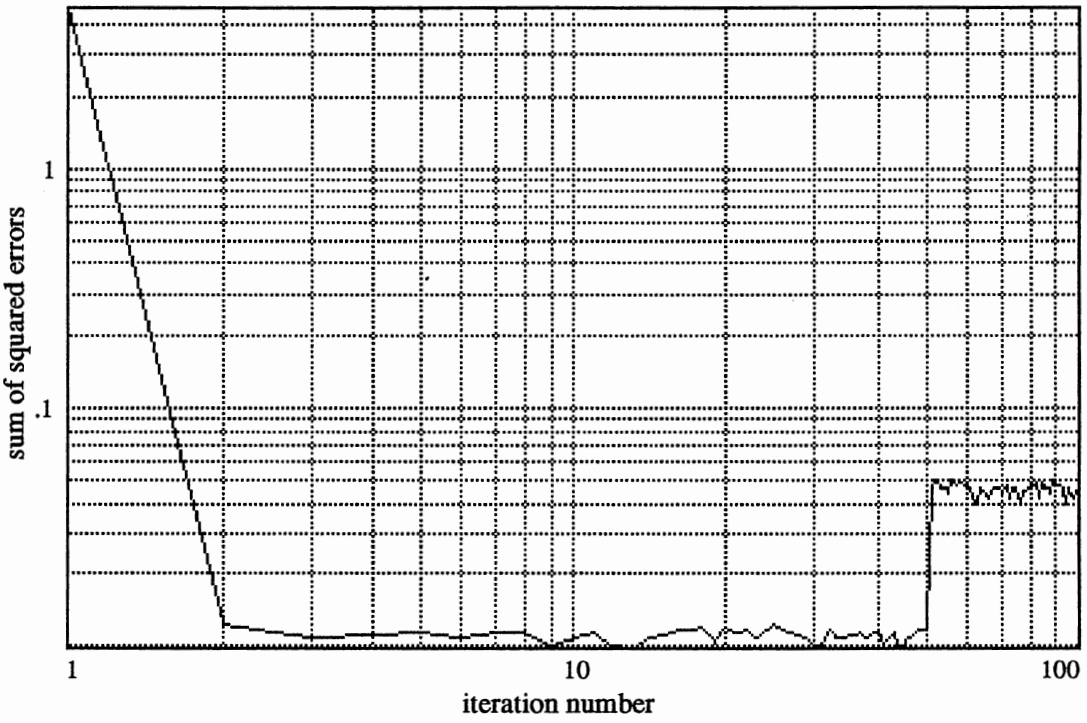
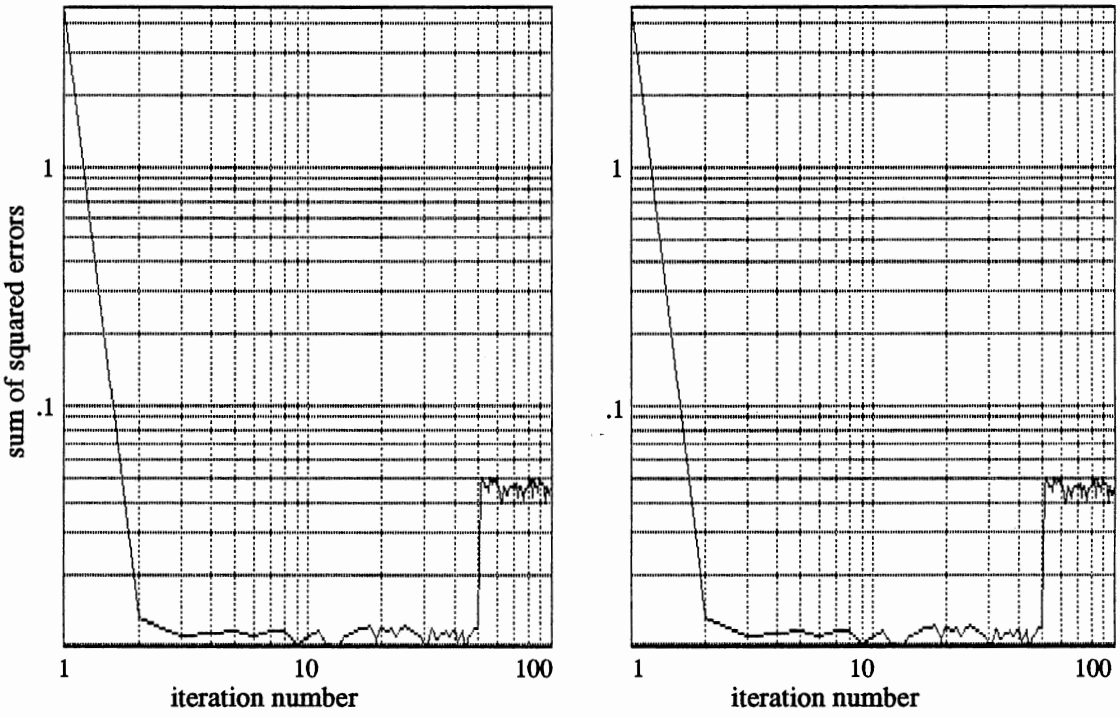


Figure 5.19. Learning Curve for Controller (case II)

Figure 5.19 shows the learning of the controller when case II occurs, the gun hits a limit but the final position is taken before the stops. Figure 5.20 shows the learning curve for the controller when case I and III occur. Case I, shown in Figure 5.20. a), occurs when the gun hits a limit and the final position is taken at the stops. Case III, shown in Figure 5.20 b), occurs when the gun hits a limit and the final position is taken after the stops.



a) Case I b) Case III

Figure 5.20. Learning Curve for Controller (case I and case III)

The training curves above were obtained when the initial positions of the gun were in the interval [1.35, 1.77], a small range. This range was slowly increased so the controller would learn from easier to harder tasks. This training, when the controller gradually learns to stabilize the plant, is shown in Figure 5.21. Here case II is considered; when the gun hits a limit the final position is taken before the stops.

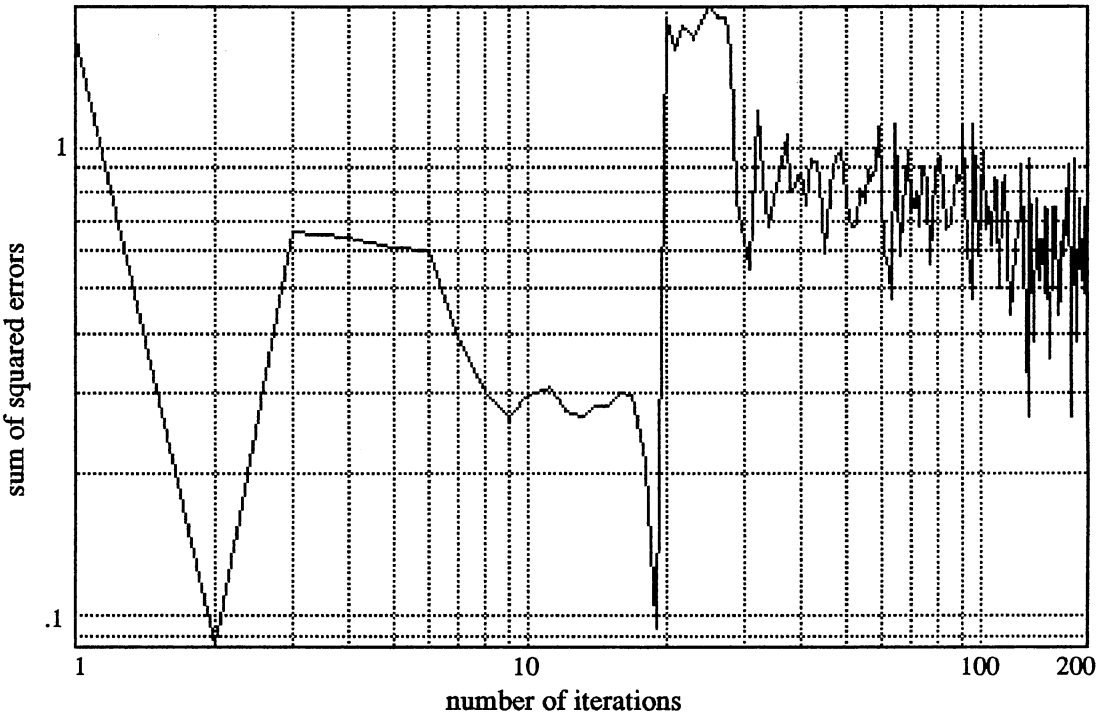


Figure 5.21. Learning Curve for Controller

First, the initial positions of the gun were kept in the interval [1.37, 1.77], but the gun was only allowed to move for 0.2 sec or when the stops were hit. Then, at the third iteration the range of the initial positions increased to [0.94, 2.20] and the gun was permitted to move for 0.3 sec. Finally, at the 20th iteration, the initial positions were in the interval [0.16, 2.99], a bigger range. At this point the gun was allowed to move for only 0.35 sec. From there on, the range of the initial positions remained fixed, but the gun could gradually move further. In the last iterations the gun moved forward for 4 sec or until stops were hit.

The controller obtained was used to control the true plant (elevation angle only). Figure 5.9 shows the operational mode of the system with the neural network controller and the saturation function. The response of the system (x_1 and x_2) for an initial condition of $[0.5 \quad 0]^T$ is given in Figure 5.22. The control input is u, dx_2/dt is the acceleration and r is the final desired position.

The controller was able to stabilize the plant, even for a large initial position, but the steady state position is offset from the desired position of $\pi/2$. This leads one to believe that the bias of the output neuron has not been fully learned. By adjusting this bias the responses of the system (position only) for three different initial positions (0.5, 1.7 and 3.0) were obtained as illustrated in Figure 5.23. All positions settle to the desired value of $\pi/2$. This shows the robustness of the controller to initial conditions.

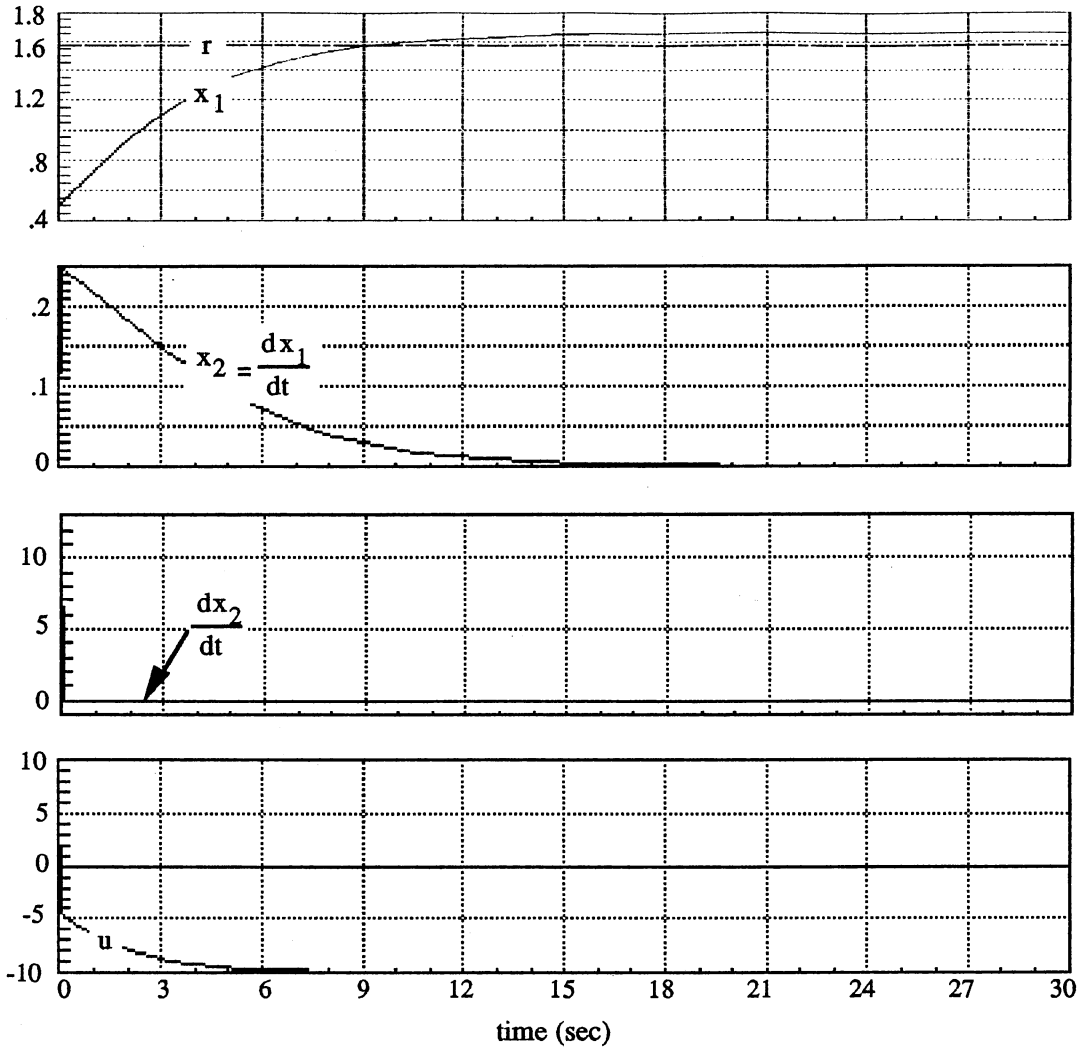


Figure 5.22 Response for Initial Condition $[0.5 \quad 0]^T$

State Variable Feedback Controller

For comparison purposes a linear controller was used on the elevation model. The linear controller is a state variable feedback controller. It is designed such that if it were applied to the linearized elevation model the response of the system to an initial

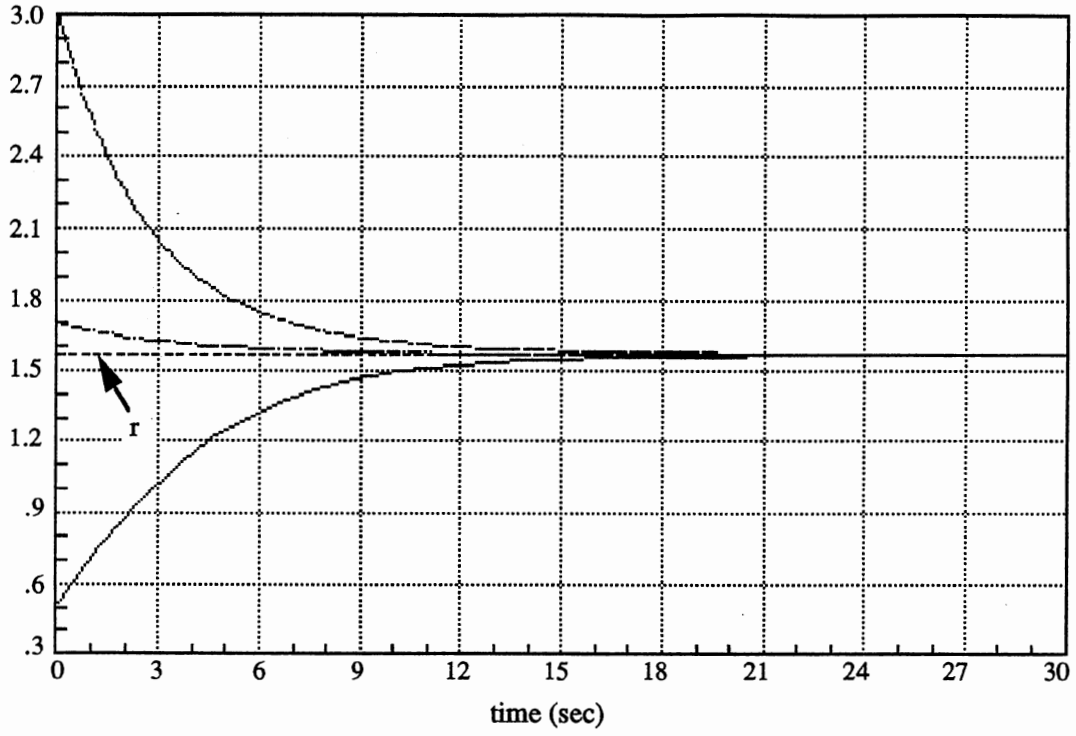


Figure 5.23. Response for Initial Positions 3.0, 1.7 and 0.5 Radians

condition close to the linearization point would roughly give the same settling time as would the system with the neural network controller.

The linearized elevation model about a position of $\pi/2$ and a velocity of zero is

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 - 2*x_2 + u$$

or

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 0 \\ 10 \end{bmatrix} = Ax + Bu + D \qquad \text{(V.6)}$$

Figure 5.24 shows the response (position only) of the neural network controller to an initial condition of $[1.7 \quad 0]^T$. The reference position, r, and 5% of the error to the steady state position is also shown. The settling time for this case is about 9 seconds with no oscillation. This requires $\xi = 1$ and $\xi * \omega_n = 3/9 = 0.33$ ($t_s = 3/\xi\omega_n$). Therefore the two poles of the closed loop system should be located at $-\xi\omega_n = -0.33$.

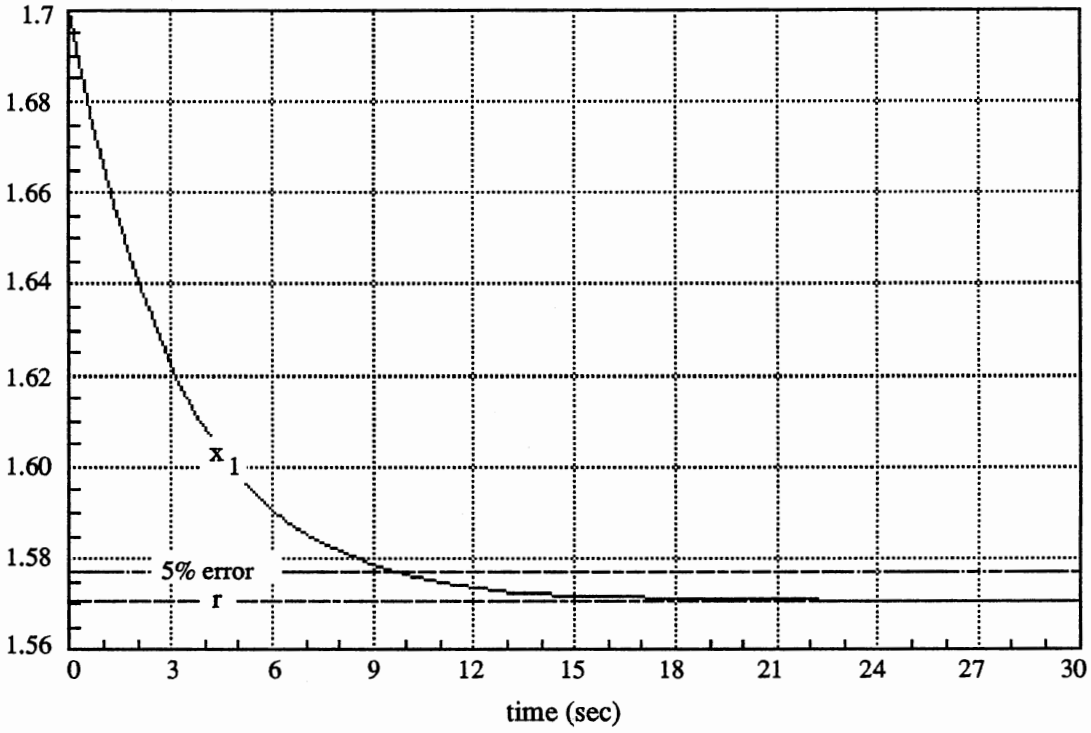


Figure 5.24. Response for Initial Condition $[1.7 \quad 0]^T$

The form of the state variable feedback controller is

$$u = -(F*x + H*r + G) \qquad (V.7)$$

Substituting the above equation in equation V.6 gives

$$\begin{aligned} dx/dt &= Ax - B(Fx + Hr + G) + D \\ &= (A - BF)x - BHr - BG + D \end{aligned} \qquad (V.8)$$

The eigenvalues of the closed loop system described by equation V.8 should be located at -0.33, -0.33. Therefore

$$F = [0.11 \quad 1.33], \qquad H = -0.11 \qquad \text{and} \qquad G = 10 \qquad (V.9)$$

Then the appropriate controller is of the form

$$u = -([0.11 \quad 1.33]*x - 0.11*r + 10) \qquad (V.10)$$

Figure 5.25 shows the response (position only) of the neural network controller (x_1) and the state variable feedback controller (x_v) for an initial condition of $[1.7 \quad 0]^T$. As one can see, the linear controller response is slower than the neural network controller. However, when the initial position was increased to 0.5 radians the state variable controller was not able to stabilize the gun, as illustrated in Figure 5.26. The stop at zero was hit in less than 1 sec. It is also shown that the neural network is capable of controlling the gun.

In the next chapter Narendra's method will be applied to the azimuth and the elevation models.

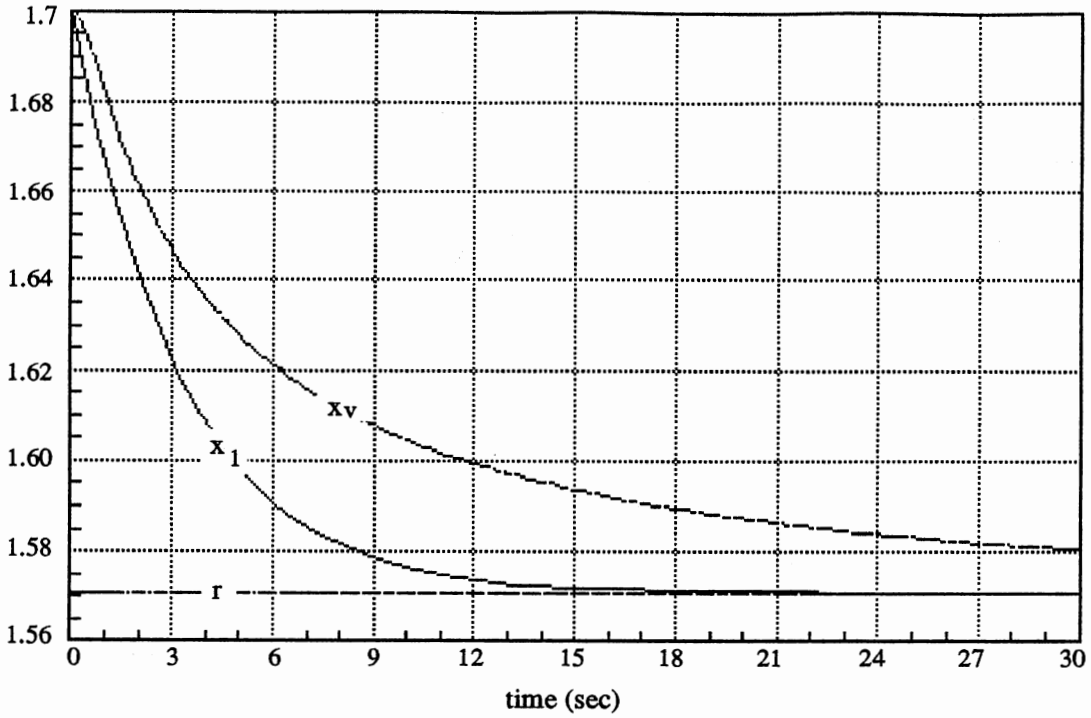


Figure 5.25. Comparison of Controllers for Initial Condition $[1.7 \quad 0]^T$

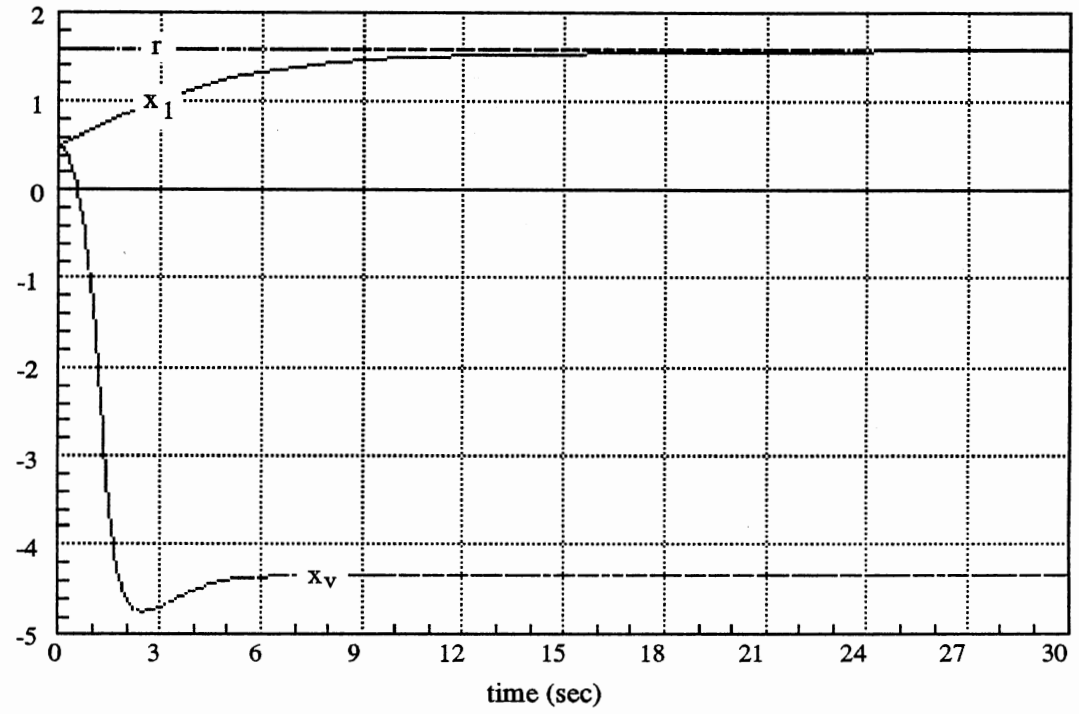


Figure 5.26. Comparison of Controllers for Initial Condition $[0.5 \quad 0]^T$

CHAPTER VI

NARENDRA'S METHOD APPLIED TO THE EXTENDED RANGE GUN

In the previous chapter neural network controllers for the models of the Extended Range Gun (ERG) were designed using Widrow's method. A linear model and nonlinear model of the system were considered. In this chapter these same models are again examined. Here the design of the neural network controller is done using Narendra's method (see Chapter IV).

Feedback Linearization

In this first approach for developing a neural network controller for the ERG (plant), all states are considered measurable. It is also assumed that there is enough information about the plant that one can describe it as belonging to model (ii) of Narendra's classification. Model (ii) assumes that states and inputs are separable functions, in particular,

$$y(k+1) = f(y(k), \ldots , y(k-n)) + u(k) \qquad \text{(VI.1)}$$

The model is also considered to be in controllable canonical form where its dynamics are given by

$$x_1(k+1) = x_2(k)$$

...

$$x_{n-1}(k+1) = x_n(k)$$

$$x_n(k+1) = f(x(k)) + u(k) \qquad \text{(VI.2)}$$

where $x(k) = [x_1(k) \quad x_2(k) \quad \ldots \quad x_n(k)]^T$ and $y(k) = x(k)$

Feedback linearization can easily be applied to a model given in this form. The idea of feedback linearization is to cancel the nonlinearities in a nonlinear system so that the closed loop dynamics will be linear.

Define the control signal as

$$u(k) = -s^T x(k) - f(x(k)) + b*r(k) \qquad \text{(VI.3)}$$

where s is a vector of constants. Substituting this equation into equation VI.2 the closed loop dynamics can be obtained as

$$x_1(k+1) = x_2(k)$$

...

$$x_{n-1}(k+1) = x_n(k)$$

$$x_n(k+1) = -s^T x(k) + b*r(k) \qquad \text{(VI.4)}$$

and the nonlinearity is cancelled. Therefore, if a neural network N_f can be obtained to approximate the nonlinearity, f(x(k)), of the plant then this nonlinearity is nullified. The operational mode of the system is shown in Figure 6.1 where z^{-1} represents a delay.

By choosing s appropriately a closed loop system with desired dynamics (a reference model) can be obtained. Model reference control is the method used by Narendra. The reference model determines how the output of the plant behaves. The relative order of the reference model should be at least equal to the relative

order of the plant (in this case, 2). Therefore, the reference model defined below was chosen as a second order system.

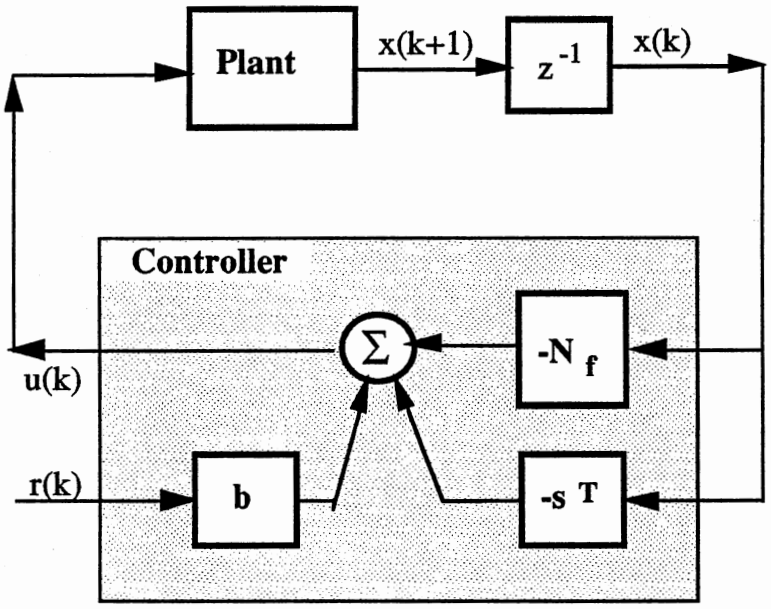


Figure 6.1. Operational Mode of the System

Reference Model

The second order reference model used here was chosen to obtain a response with a settling time (t_s) of 1 sec and no oscillations. This required $\xi = 1$ and $\xi * \omega_n = 3/1 = 3$ ($t_s = 3/\xi\omega_n$). Consequently, $\omega_n = 3$.

The output of the reference model and the output of the plant should describe the same quantity. The outputs of the plant are position and velocity. When a step input is applied to the reference

model it is desired that the steady state value of the output position settles to the value of the reference input and the output velocity goes to zero. This results in the following transfer function for the reference model between position and input

$$\frac{\omega_n^2}{s^2+2\xi\omega_n s+\omega_n^2}=\frac{3^2}{s^2+2(3)s+3^2}=\frac{9}{s^2+6s+9} \qquad \text{(VI.5)}$$

Velocity is the other output of the plant. If one of the state variables is position then the reference model is subject to the constraint that $x_{m2} = dx_{m1}/dt$ where x_{m1} is the state variable position and x_{m2} is the state variable velocity. This results in the state variable form of the reference model as shown below

$$\begin{bmatrix} \frac{dx_{m1}}{dt} \\ \frac{dx_{m2}}{dt} \end{bmatrix}=\begin{bmatrix} 0 & 1 \\ -9 & -6 \end{bmatrix}\begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix}+\begin{bmatrix} 0 \\ 9 \end{bmatrix}r \qquad \text{(VI.6)}$$

Discretizing with a sampling interval of 0.01 sec and using Euler's integration method, the following equation is obtained for the discrete time system

$$x_m(k+1)=\begin{bmatrix} 1.0 & 0.01 \\ -0.09 & 0.94 \end{bmatrix}x_m(k)+\begin{bmatrix} 0.0 \\ 0.09 \end{bmatrix}r(k) \qquad \text{(VI.7)}$$

where $x_m(k) = [x_{m1}(k) \quad x_{m2}(k)]^T$ are the states of the reference model and r(k) is the reference input.

Azimuth Model

The azimuth model (θ fixed) is shown in Figure 5.3. It is a second order linear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$

$$dx_2/dt = (-\ 2\ x_2\ +\ u)/\sin^2(\theta) \qquad \text{(VI.8)}$$

where: $x_1 = \phi$

$x_2 = d\phi/dt$

u - input current to the motor that moves the gun in ϕ

In this study θ is fixed at $\pi/2$ (90^o) and Euler's integration method where $x(k+1) = x(k) + \Delta t*dx/dt$ is used to obtain x_1 and x_2 with $\Delta t = 0.01$ sec, the sampling interval. Therefore,

$$x_1(k+1) = x_1(k) + \Delta t*x_2(k)$$

$$x_2(k+1) = x_2(k) - 2*\Delta t*x_2(k) + \Delta t*u(k) = f(x(k)) + \Delta t*u(k) \qquad \text{(VI.9)}$$

As described above, feedback linearization requires plant identification. Figure 4.11 shows a sketch of the procedure for plant identification.

With this particular linear plant, if the control input u is set to zero $y(k+1) = f(x(k)) = N_f[x(k)] = N[x(k),0]$ where N[ ] is the neural network plant identified earlier in Chapter V (Figure 5.5). The neural network plant has a single layer, with 3 inputs which correspond to the 2 state variables and the input current, and 2 outputs which correspond to the 2 state variables as shown in Figure 5.4. All neurons have a linear transfer function, $f(x) = x$. Here only the second output (x_2) will be used.

The control input is then

$$u(k) = (1/.01)*(-[0.09\quad -0.94]*x(k) - N_f[x(k)] + 0.09*r(k)) \qquad \text{(VI.10)}$$

This controller was used to control the true plant. Figure 6.1 illustrates the operational mode of the system. Figure 6.2 shows the response of the system and the reference model to an initial condition $[3\quad 0]^T$ when a constant reference input, r(k), of zero is

applied. The response of the system (x_1 and x_2) is identical to that of the reference model (x_{m1} and x_{m2}). The controller's output is u and dx_2/dt is the acceleration of the plant.

Elevation Model

The elevation model (ϕ fixed) is shown in Figure 5.11. It is a second order nonlinear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 \sin(x_1) - 2 x_2 + u \qquad \text{(VI.11)}$$

where: $x_1 = \theta$

$x_2 = d\theta/dt$

u - input current to the motor that moves the gun in θ

The integration method used to obtain x_1 and x_2 was Euler's method where $x(k+1) = x(k) + \Delta t*dx/dt$ with $\Delta t = 0.01$ sec. Therefore,

$$x_1(k+1) = x_1(k) + \Delta t*x_2(k)$$

$$x_2(k+1) = 10*\Delta t*\sin(x_1) + (1 - 2*\Delta t)*x_2(k) + \Delta t*u(k)$$

$$= f(x(k)) + \Delta t*u(k) \qquad \text{(VI.12)}$$

As described earlier, feedback linearization requires plant identification. The conjugate gradient algorithm with line search was the only algorithm employed to identify the nonlinearity, f(x(k)), in the plant.

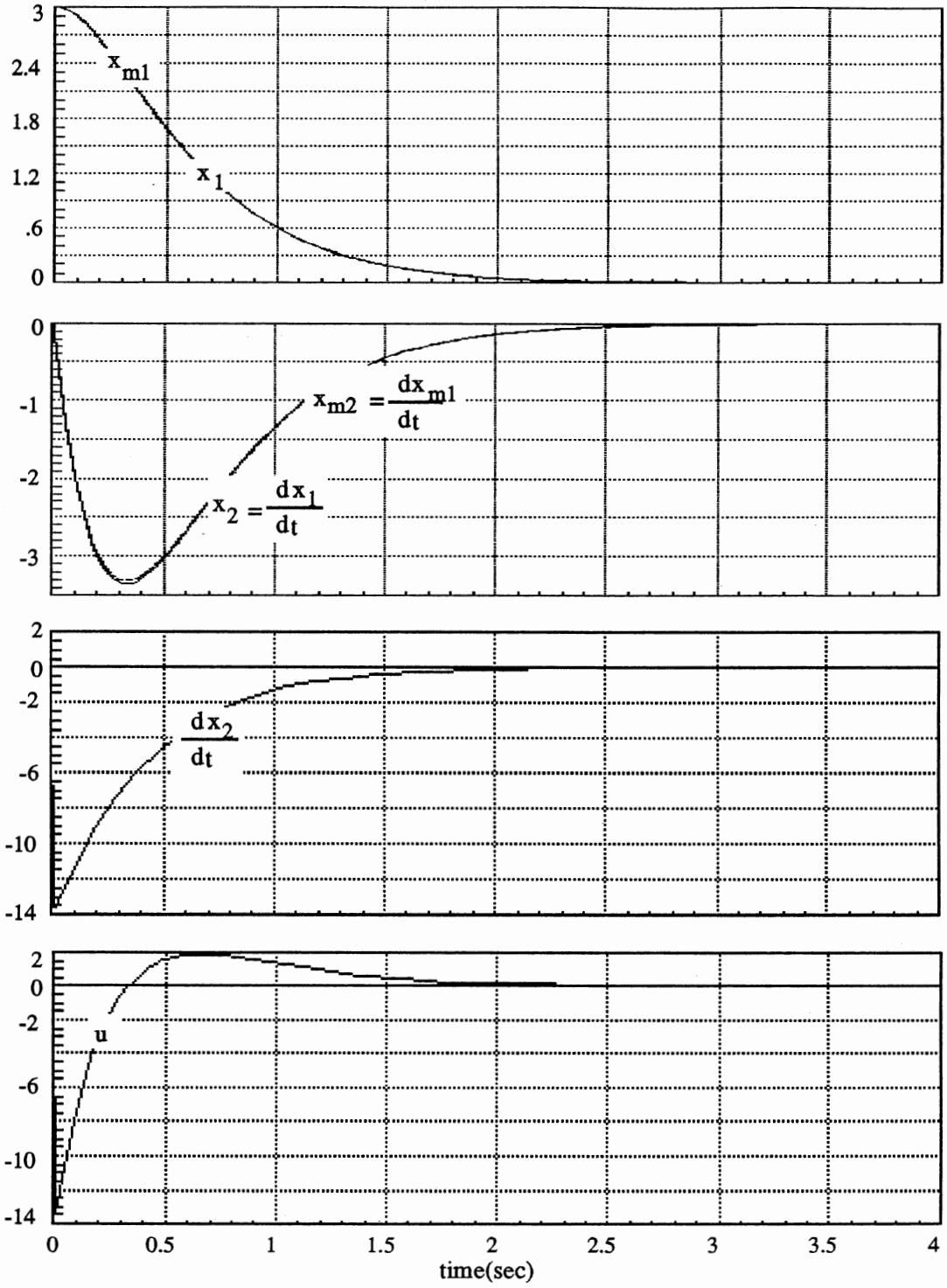


Figure 6.2. Response for Initial Condition $[3 \quad 0]^T$

The neural network N_f has two layers, with 12 neurons in the hidden layer. It has two inputs, corresponding to the two state variables, and one output, corresponding to $x_2(k)$. The hidden neurons have a sigmoid type transfer function, and the output neurons have a linear transfer function.

N_f is trained offline. During the training process, states are generated randomly. These are presented to the neural network and the true plant to obtain the predicted and the desired next state. An error is calculated and backpropagated allowing the weights of the neural network to be adjusted. This is done for many iterations until a performance criterion is satisfied. The learning curve for N_f is shown in Figure 6.3.

The neural network N_f is then used as part of the controller, where the control input u is given by equation VI.10. The operational mode of the system is illustrated in Figure 6.1. Figure 6.4 shows the response of the system (x_1 and x_2) and the reference model (x_{m1} and x_{m2}) to an initial condition $[0.5 \quad 0]^T$ when a constant reference input (x_d), r(k), of $\pi/2$ is applied. It also shows the controlled input u to the plant and the acceleration of the plant, dx_2/dt. The response of the system is identical to that of the reference model.

State Variable Feedback Controller

For comparison purposes a linear controller was used on the elevation model. The linear controller is a state variable feedback

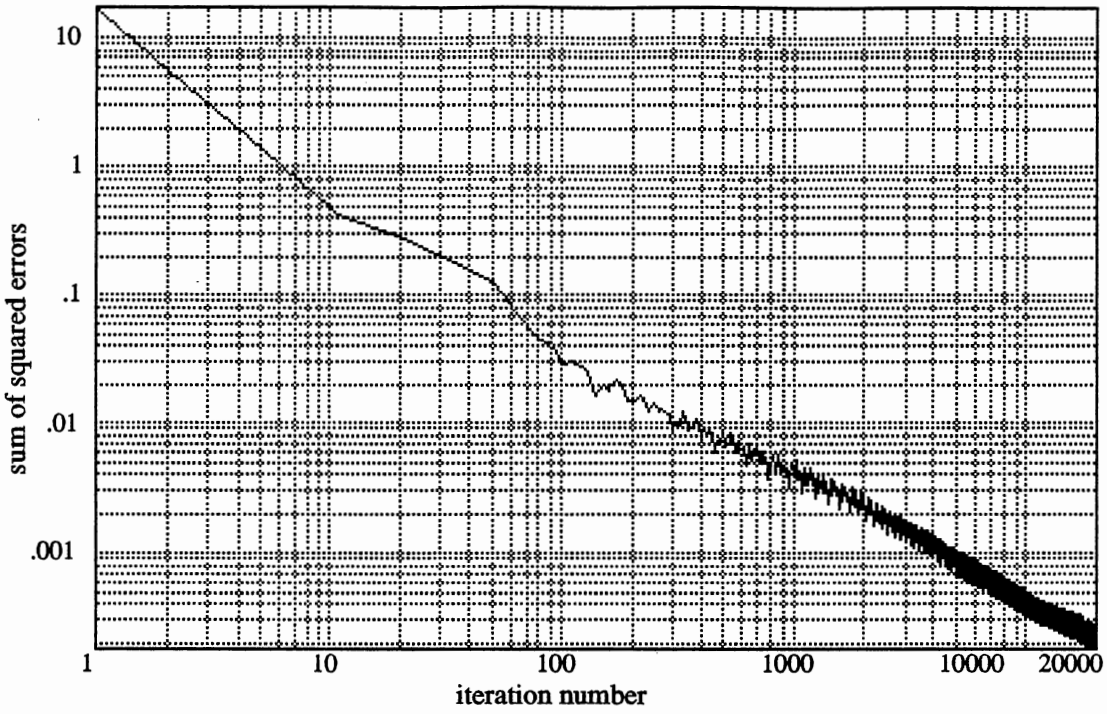


Figure 6.3. Learning Curve for Neural Network N_f

controller. It is designed such that if it were applied to the linearized elevation model the closed loop system would respond like the reference model given above.

The linearized elevation model about a position of $\pi/2$ and a velocity of zero is

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 - 2*x_2 + u$$

or

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 0 \\ 10 \end{bmatrix} = Ax + Bu + D \qquad \text{(VI.13)}$$

The form of the state variable feedback controller is

$$u = -(F*x + H*r + G) \qquad \text{(VI.14)}$$

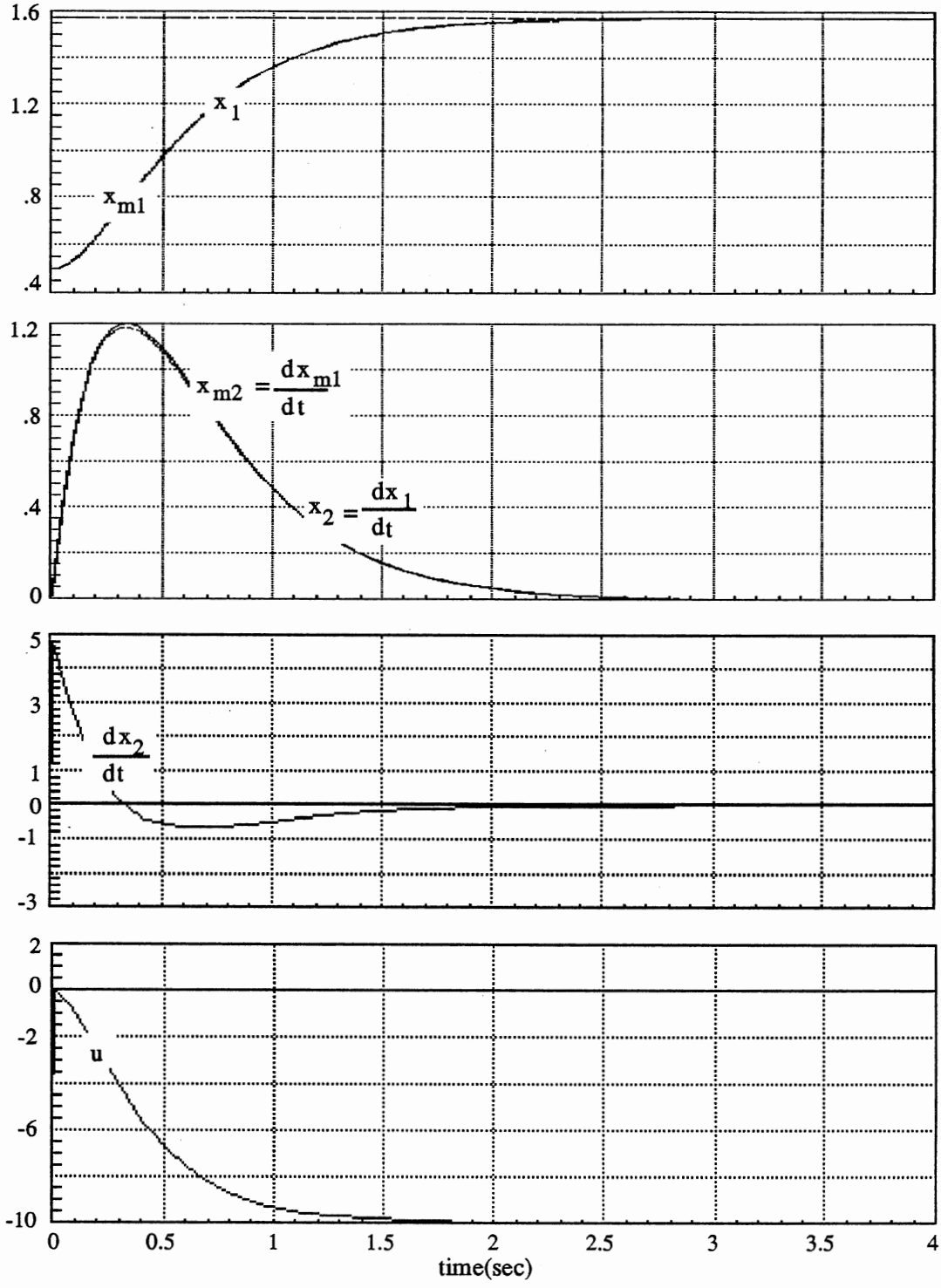


Figure 6.4. Response for Initial Condition $[0.5 \quad 0]^T$

Substituting the above equation in equation VI.13 gives

$$dx/dt = Ax - B(Fx + Hr + G) + D$$

$$= (A - BF)x - BHr - BG + D \qquad \text{(VI.15)}$$

If F, H and G are chosen as

$$F = [9 \;\; 4], \qquad H = -9 \qquad \text{and} \qquad G = 10 \qquad \text{(VI.16)}$$

Then the appropriate controller is of the form

$$u = -([9 \;\; 4]*x - 9*r + 10) \qquad \text{(VI.17)}$$

And perfect model following is achieved.

Figure 6.5 shows the response (position only) of the neural network controller (x_1), the reference model (x_{m1}), the reference position ($r = \pi/2$) and the state variable feedback controller (x_{c1}) for an initial condition of $[0.5 \;\; 0]^T$. As one can see, all positions go to the reference input but the linear controller response is slower than the neural network controller and the reference model.

General Method

In the above discussion all states of the plant are measurable. But it is common to find that not all the states are assessable. In this case Narendra's general method is applied and the plant can be described as belonging to model (iv).

Model (iv) assumes that states and inputs are not separable functions. The equation describing this general model is

$$y(k+1) = f(y(k), \ldots, y(k-n), u(k), \ldots, u(k-n)) \qquad \text{(VI.18)}$$

Here it will be considered that the only measurable state is position.

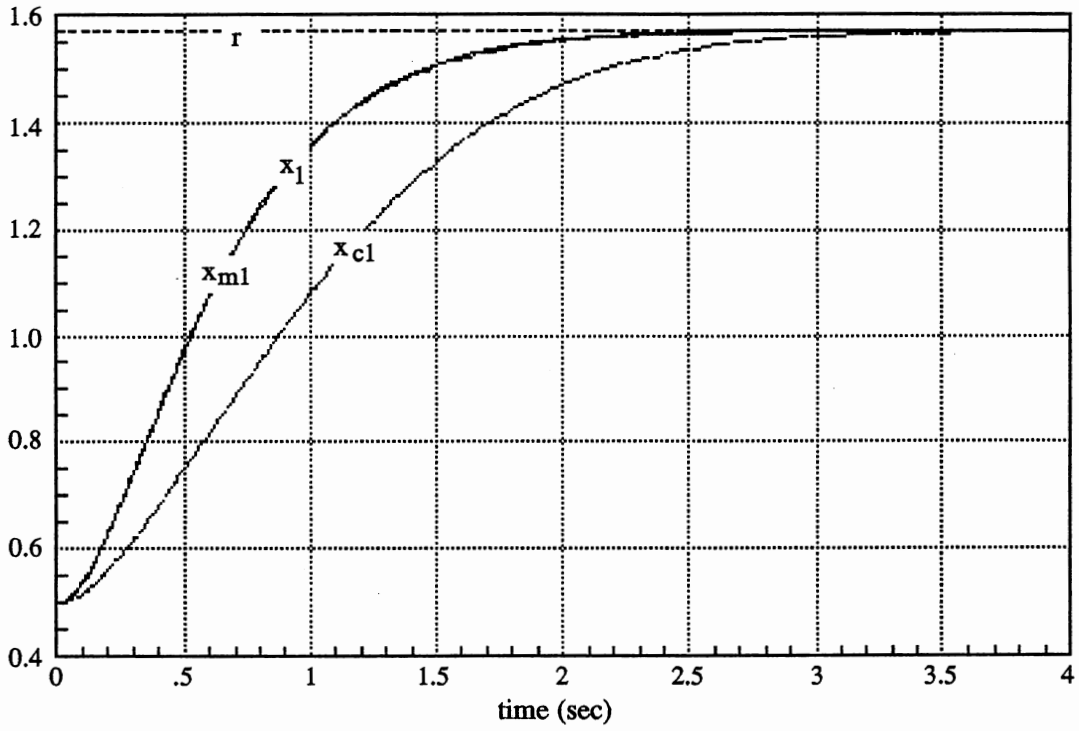


Figure 6.5. Comparison of Controllers

A neural network is then trained to imitate the plant. During the training process, inputs (u(k)) and initial positions (y(0)) are generated randomly. All other states are set to zero. Past values of y are set to y(0) and past values of u(k) are set to zero. These are presented to the neural network and the true plant. The plant is then allowed to move forward for m steps, or until stops are hit. At this point, new random initial positions are generated and the system moves forward once more. Inputs are random at each step. This is done for many iterations until a performance criterion is satisfied. Figure 6.6 shows a schematic diagram of the training of the plant.

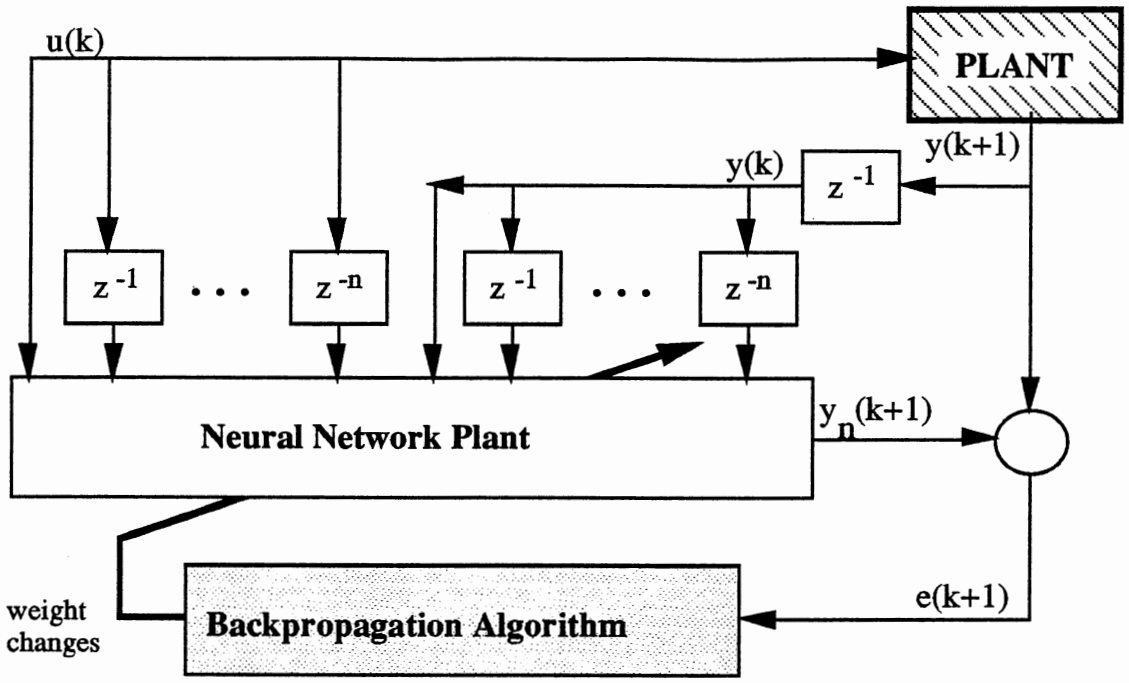


Figure 6.6. Plant Training

After the neural network plant has learned the true plant's transfer function it is used in training the controller. The controller will be trained to give the correct input u that will drive the plant in accordance with a reference model. The neural network plant is used to backpropagate the error ($y_m(k+1)$ - $y(k+1)$) to adjust the weights of the controller.

The equation that describes the controller is

$$u(k) = N_c[y(k), \ldots, y(k-n), u(k-1), \ldots, u(k-n), r(k), \ldots, r(k-n)] \quad \text{(VI.19)}$$

where N_c is the neural network obtained through the training process depicted below.

During the controller training process initial positions (y(0)) and reference inputs (r(k)) are generated randomly. All other

states are set to zero. Past values of y are set to y(0) and past values of u(k) are set to zero. The system (controller and plant) is then allowed to move forward for m steps, or until stops are hit. At this point, new random initial positions are generated and the system moves forward once more. After each step an error is calculated and backpropagated so that the weights of the neural network controller are adjusted. This is done for many iterations until a performance criterion is satisfied. Figure 6.7 shows the schematic for training the controller. This neural network controller is then used in the operational mode of the system as shown in Figure 6.8.

Second Order Models

In this section the above method is applied to develop a neural network controller for the previous given models of the ERG. Here the only measurable state is position (x_1). It is assumed that the order of the system is known, in this case 2. Therefore, the neural network plant is

$$y(k+1) = N_p[y(k),\ y(k-1),\ u(k),\ u(k-1)] \qquad \text{(VI.20)}$$

and the neural network controller is

$$u(k) = N_c[r(k),\ r(k-1),\ y(k),\ y(k-1),\ u(k-1)] \qquad \text{(VI.21)}$$

Next, the reference model is defined.

Reference Model

The reference model should have the same relative order as the plant. Therefore, a second order system was chosen. This system is

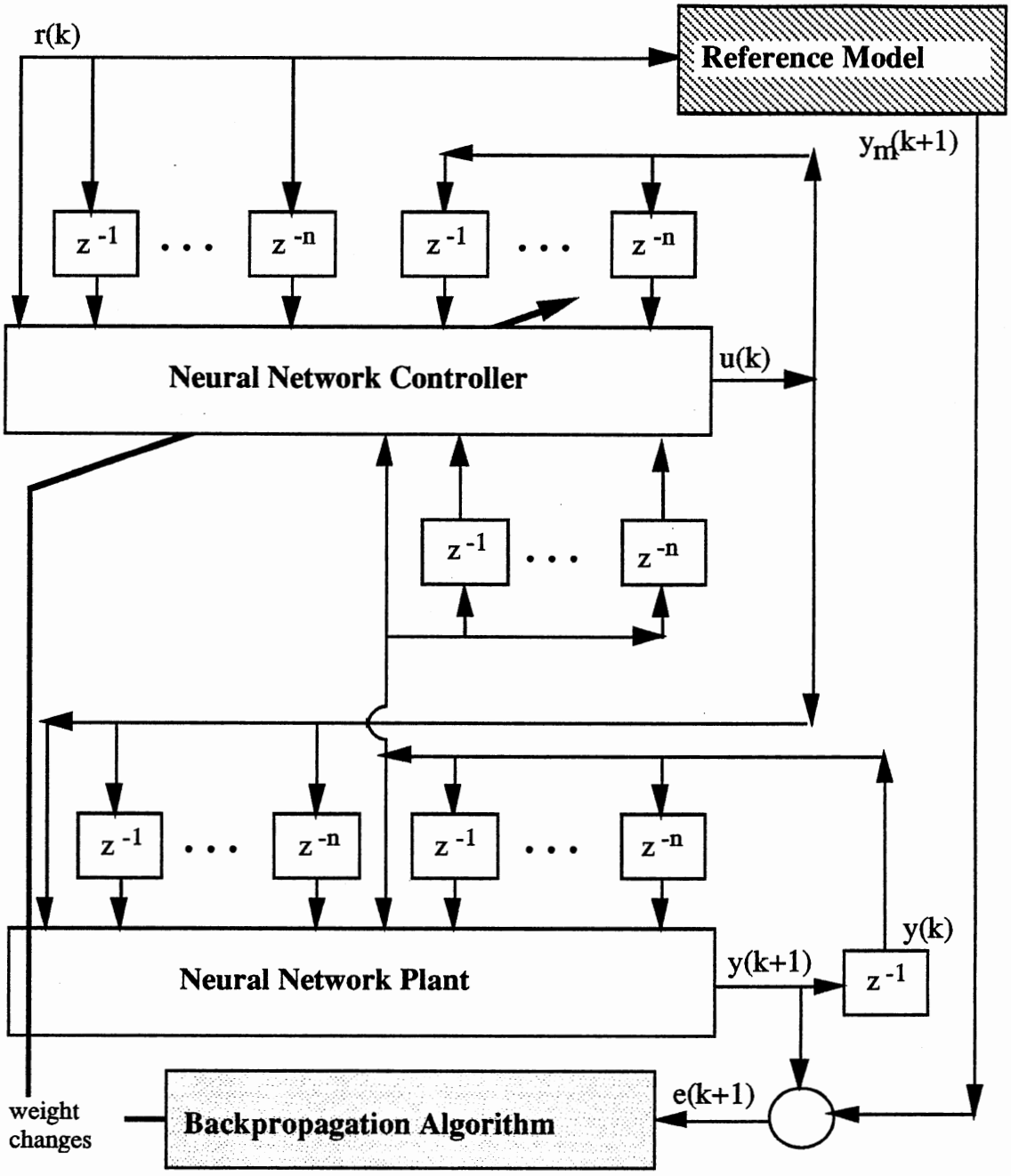


Figure 6.7. Controller Training

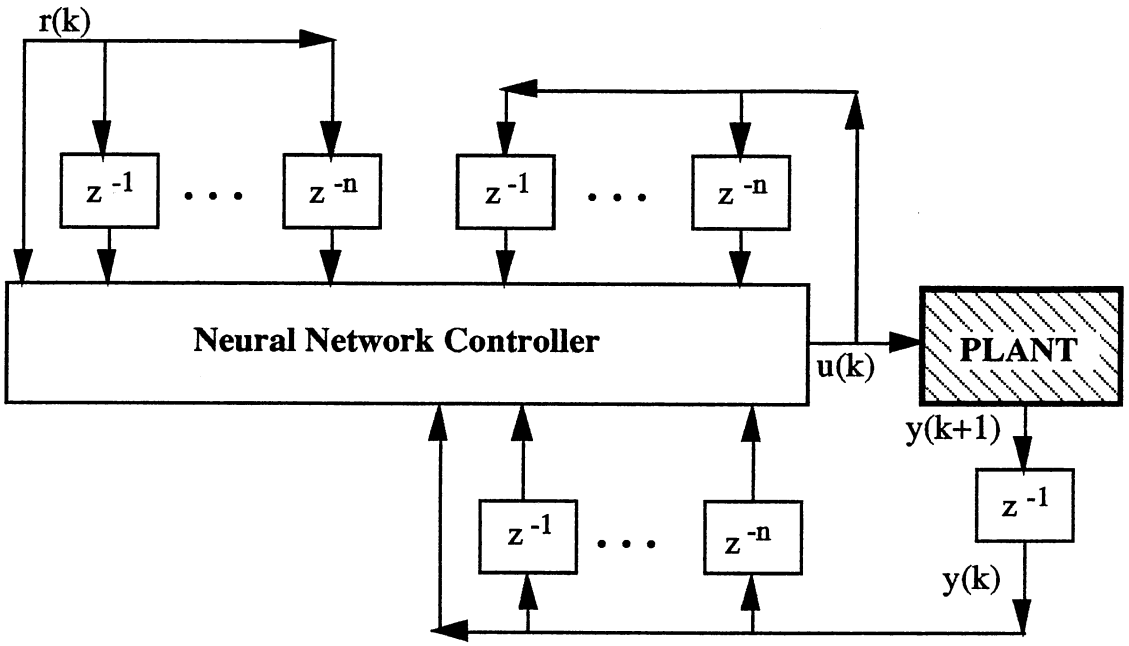


Figure 6.8. Operational Mode of the System

the same one described in a section above. The state variable form of the reference model is

$$\begin{bmatrix} \frac{dx_{m1}}{dt} \\ \frac{dx_{m2}}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -9 & -6 \end{bmatrix} \begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix} + \begin{bmatrix} 0 \\ 9 \end{bmatrix} r \tag{VI.22}$$

Discretizing with a sampling interval of 0.01 sec the following equation is obtained

$$x_m(k+1) = \begin{bmatrix} 0.9996 & 0.0097 \\ -0.08734 & 0.09413 \end{bmatrix} x_m(k) + \begin{bmatrix} 0.0004 \\ 0.08734 \end{bmatrix} r(k) \tag{VI.23}$$

where $x_m(k) = [x_{m1}(k) \quad x_{m2}(k)]^T$ are the states of the reference model and r(k) is the reference input.

The following transfer function can be obtained between position, $y_m(k)$, and the reference input, r(k)

$$\frac{Y_m}{R} = \frac{(4.411*z+4.3237)*.0001}{z^2-1.9409*z+0.9417} \qquad \text{(VI.24)}$$

which gives

$$y_m(k+1) = 1.9409*y_m(k) - 0.9417*y_m(k-1) + (4.411*r(k) + 4.3237*r(k-1))*.0001 \qquad \text{(VI.25)}$$

In this section a sampling interval of 0.1 sec is also used. The following discrete system is obtained

$$x_m(k+1) = \begin{bmatrix} 0.9631 & 0.0741 \\ -0.6667 & 0.5186 \end{bmatrix} x_m(k) + \begin{bmatrix} 0.0369 \\ 0.6667 \end{bmatrix} r(k) \qquad \text{(VI.26)}$$

where $x_m(k) = [x_{m1}(k) \quad x_{m2}(k)]^T$ are the states of the reference model and r(k) is the reference input. Therefore, the transfer function between position, $y_m(k) = x_{m1}(k)$, and the reference input is

$$\frac{Y_m}{R} = \frac{0.0369z+0.0302}{z^2-1.4816z+0.5488} \qquad \text{(VI.27)}$$

which results

$$y_m(k+1) = 1.4816*y_m(k) - 0.5488*y_m(k-1) + 0.0369*r(k) + 0.0302*r(k-1) \qquad \text{(VI.28)}$$

Azimuth Model

The azimuth model (θ fixed) is shown in Figure 5.3. It is a second order linear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$
$$dx_2/dt = (-2\,x_2 + u)/\sin^2(\theta) \qquad \text{(VI.29)}$$

where: $x_1 = \phi$

$x_2 = d\phi/dt$

u - input current to the motor that moves the gun in ϕ

In this study θ is fixed at $\pi/2$ (90^o) and third-order Runge-Kutta integration (see Appendix) is used to obtain x_1 and x_2 with a sampling interval of 0.1 and 0.01 sec.

The plant can be described as

$$y(k+1) = f(y(k), y(k-1), u(k), u(k-1)) \qquad \text{(VI.30)}$$

A neural network N_p with one layer of linear neurons having four inputs (y(k), y(k-1), u(k) and u(k-1)) and one output (y(k+1)) was trained to imitate the plant. First, a sampling interval of 0.1 sec was used. The learning curve for N_p is shown in Figure 6.9. This neural network was then used to define a controller for the plant.

The controller can be represented as

$$u(k) = N_c[r(k), r(k-1), y(k), y(k-1), u(k-1)] \qquad \text{(VI.31)}$$

Neural network N_c has one layer of linear neurons with five inputs and one output. It was trained to control the plant. The learning curve for N_c is shown in Figure 6.10.

This controller was then used to control the plant. Figure 6.8 shows the operational mode of the system. The response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 3.0 radians when a constant reference input of zero is applied is illustrated in Figure 6.11. The controlled input is u and dx_2/dt is the acceleration of the plant.

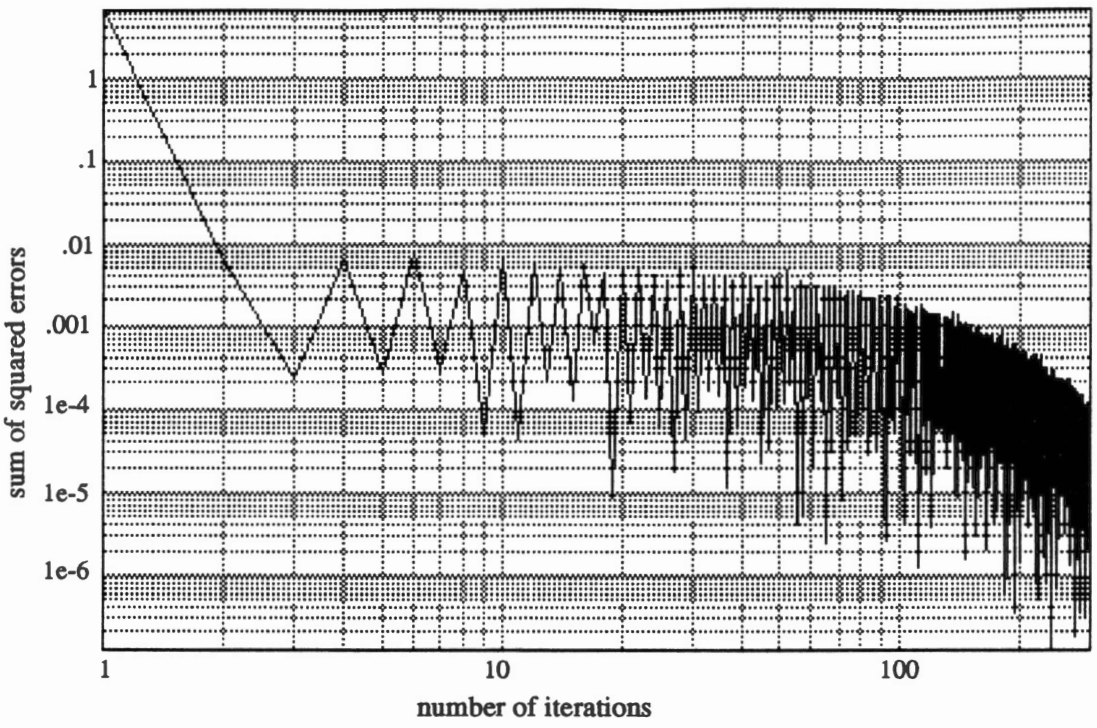


Figure 6.9. Learning Curve for Neural Network N_p (Δt=0.1)

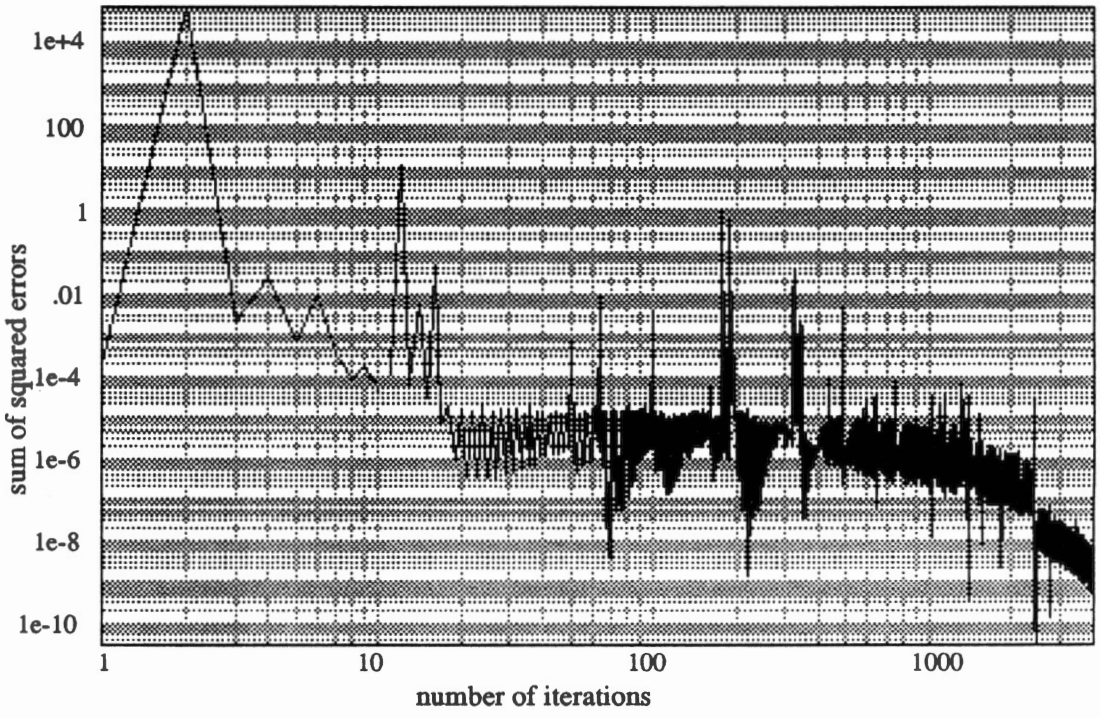


Figure 6.10. Learning Curve for Neural Network N_c (Δt=0.1)

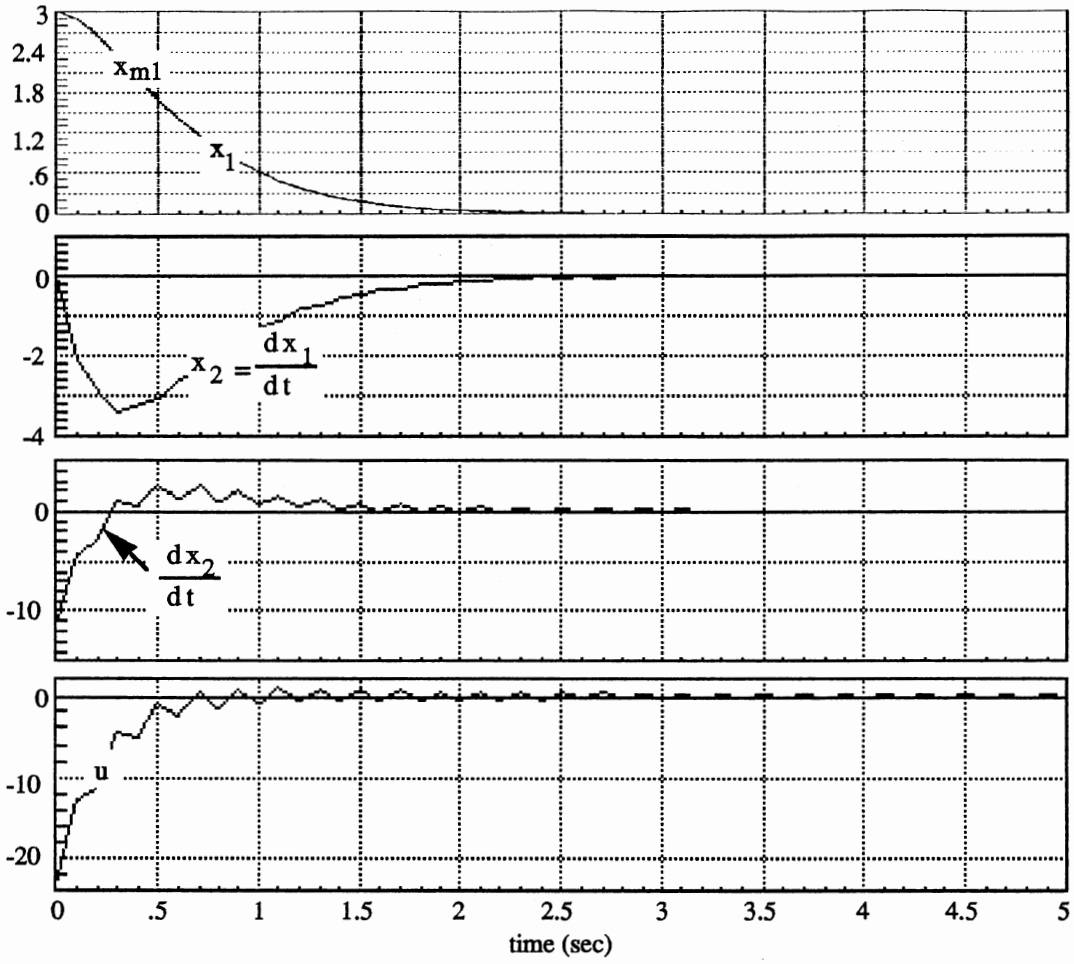


Figure 6.11. Response for Initial Position 3.0 (Δt=0.1)

Next, a sampling interval of 0.01 sec is used. The learning curve for N_p is shown in Figure 6.12. Although the error decreased to 1e-6, the weights of the plant did not reach their true value. More iterations were performed without success, the weights of the plant remained unchanged. When the sampling interval of 0.1 sec was used the final weights of the plant did converge to their true value.

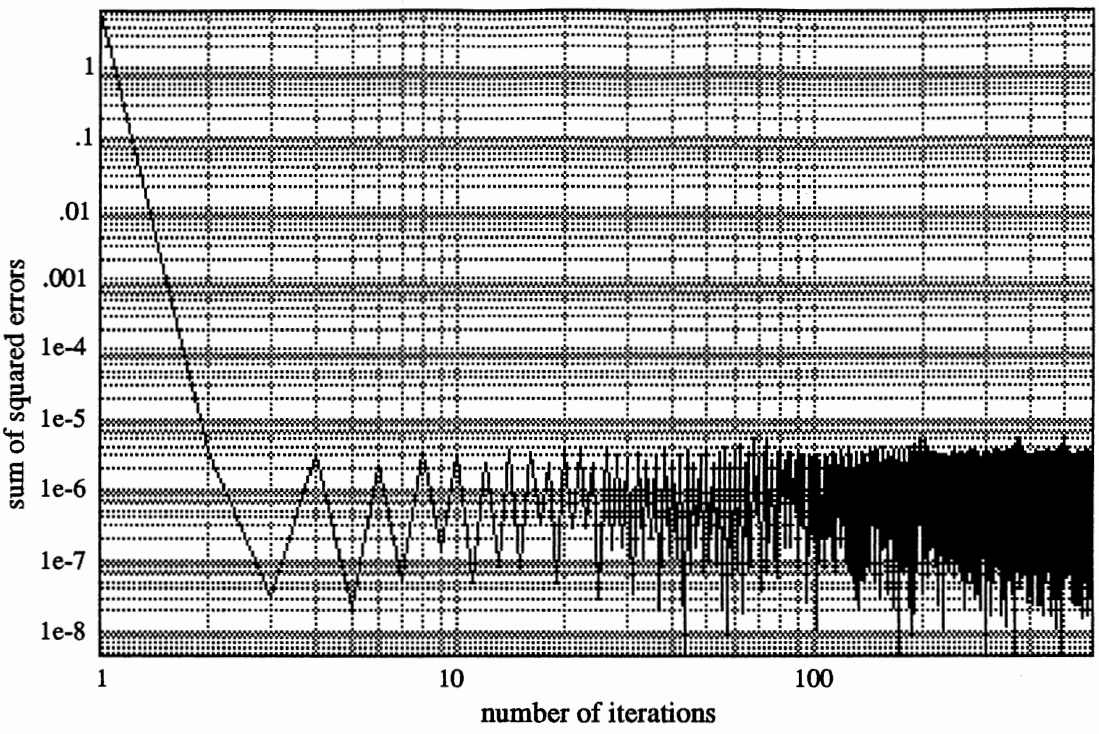


Figure 6.12. Learning Curve for Neural Network N_p (Δt=0.01)

The algorithm used to train the neural networks above was the conjugate gradient. Next, a neural network plant and controller are trained with the Levenberg-Marquardt algorithm. This agorithm is a variation of backpropagation and proved to be more efficient in the training of these networks.

The Levenberg-Marquardt algorithm is based on least squares optimization. The algorithm is derived from the Gauss-Newton method of minimization which requires the computation of the inverse of J^TJ where J is the Jacobian matrix. The J^TJ matrix can be singular, therefore its inverse may not exist. The Levenberg-Marquardt includes a technique to overcome this problem [8].

A neural network Np is trained to imitate the plant using a sampling interval of 0.01 sec. The learning curve for N_p is shown in Figure 6.13. As one can see, faster convergence is obtained. The final weights of the plant corresponded to their true values. This neural network is then used to define a controller for the plant. The learning curve for N_c is shown in Figure 6.14.

This controller was then used to control the plant. Figure 6.8 shows the operational mode of the system. The response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 3.0 radians when a constant reference input of zero is applied is illustrated in Figure 6.15. The controlled input is u and dx_2/dt is the acceleration of the plant.

Elevation Model

The elevation model (ϕ fixed) is shown in Figure 5.11. It is a second order nonlinear system. The equations in state variable form that describe this system are

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 \sin(x_1) - 2 x_2 + u \qquad \text{(VI.32)}$$

where: $x_1 = \theta$

$x_2 = d\theta/dt$

u - input current to the motor that moves the gun in θ

The integration method used to obtain x_1 and x_2 was third-order Runge-Kutta (see Appendix) with a sampling interval of 0.1 and 0.01 sec.

Figure 6.13. Learning Curve for Neural Network N_p (Δt=0.01)

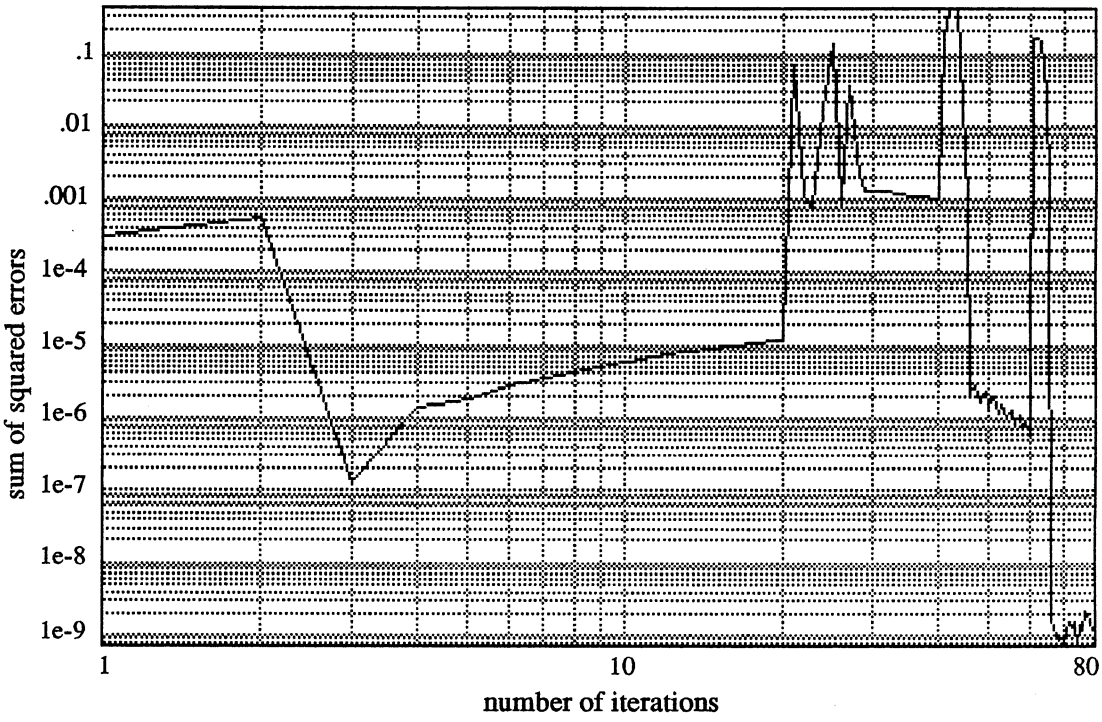


Figure 6.14. Learning Curve for Neural Network N_c (Δt=0.01)

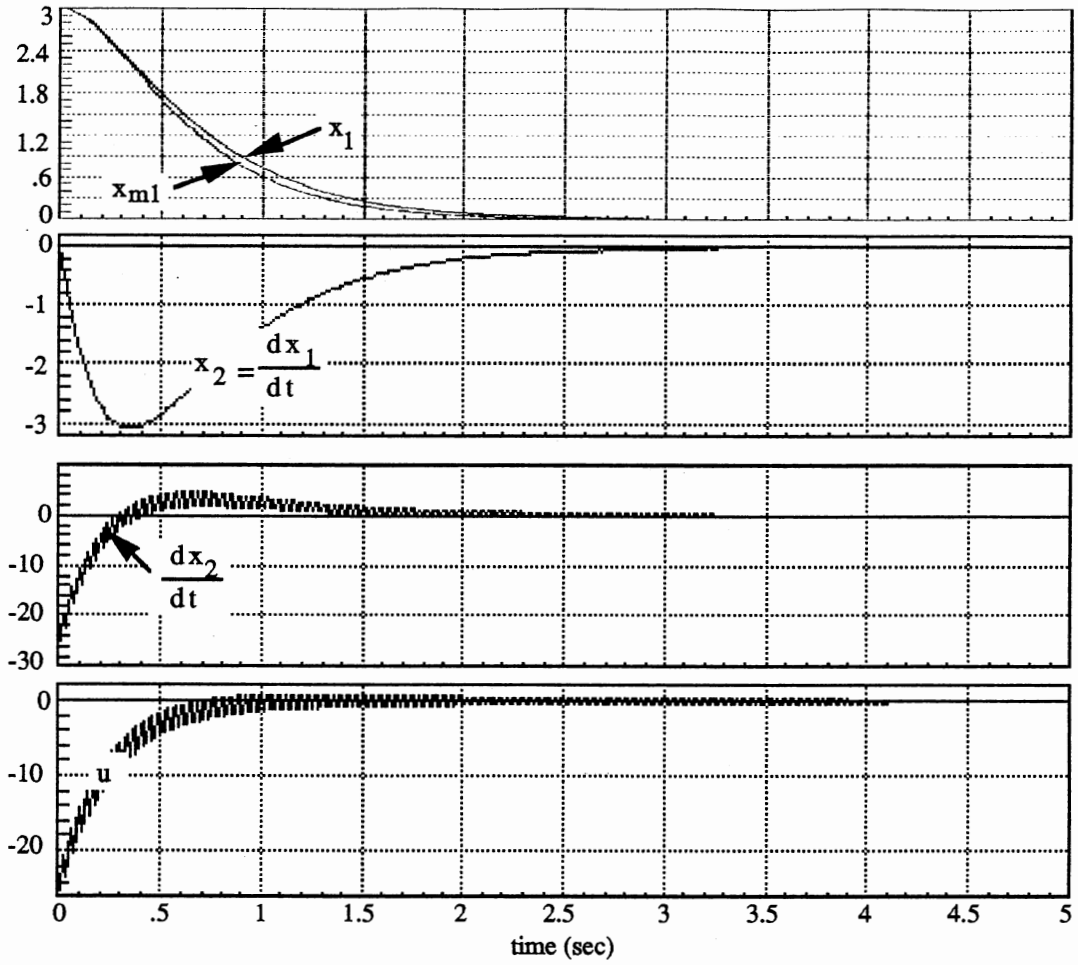


Figure 6.15. Response for Initial Position 3.0 (Δt=0.01)

The plant can be described as

$$y(k+1) = f(y(k), y(k-1), u(k), u(k-1)) \qquad \text{(VI.33)}$$

Neural network N_p has two layers, with 5 neurons in the hidden layer. It has four inputs (y(k), y(k-1), u(k) and u(k-1)) and one output (y(k+1)). The hidden neurons have a sigmoid type transfer function and the output neurons have a linear transfer function. It was trained to imitate the plant. First, a sampling interval of 0.1

sec was used. The learning curve for N_p is shown in Figure 6.16. This neural network was then used to define a controller for the plant.

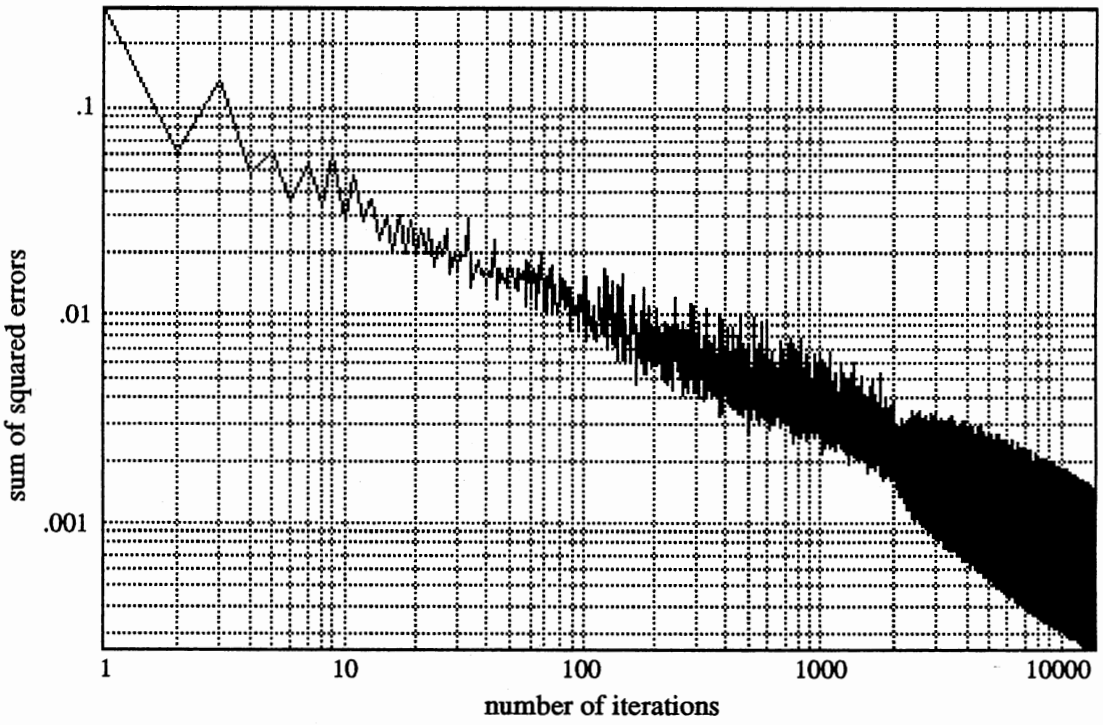


Figure 6.16. Learning Curve for Neural Network N_p (Δt=0.1)

The controller can be represented as

$$u(k) = N_c[r(k),\ r(k-1),\ y(k),\ y(k-1),\ u(k-1)] \qquad \text{(VI.34)}$$

Neural network N_c has two layers, with 12 neurons in the hidden layer. It has five inputs (r(k), r(k-1), y(k), y(k-1) and u(k-1)) and one output (u(k)). The hidden neurons have a sigmoid type

transfer function and the output neurons have a linear transfer function. It was trained to control the plant. The learning curve for N_c is shown in Figure 6.17. The weights of the plant did not change when more iterations were performed. Different initial weight values were investigated, either the same type of curve was produced or the weights became very large. Therefore, a different algorithm, the Levenberg-Marquardt algorithm, is employed in training the neural networks.

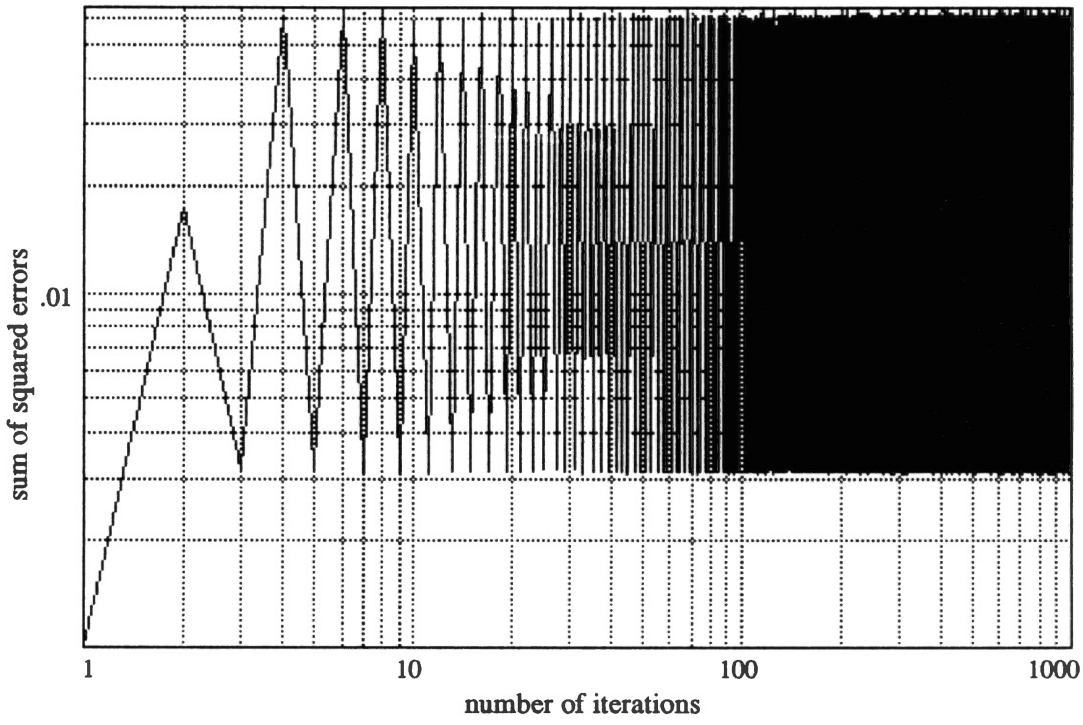


Figure 6.17. Learning Curve for Neural Network N_c (Δt=0.1)

Next, the neural network plant and controller is trained with the Levenberg-Marquardt algorithm. First, a sampling interval of 0.1 sec is used. A neural network N_p is trained to imitate the plant. The learning curve for N_p is shown in Figure 6.18. This neural network is then used to define a controller for the plant. The learning curve for N_c is shown in Figure 6.19.

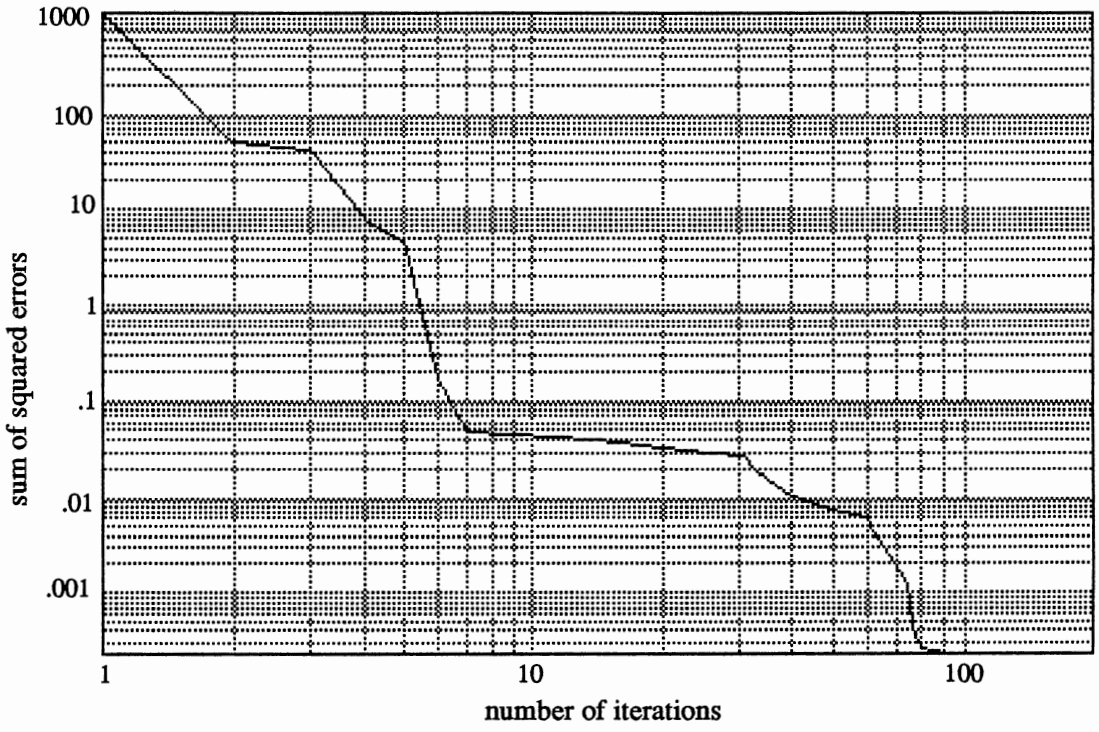


Figure 6.18. Learning Curve for Neural Network N_p (Δt=0.1)

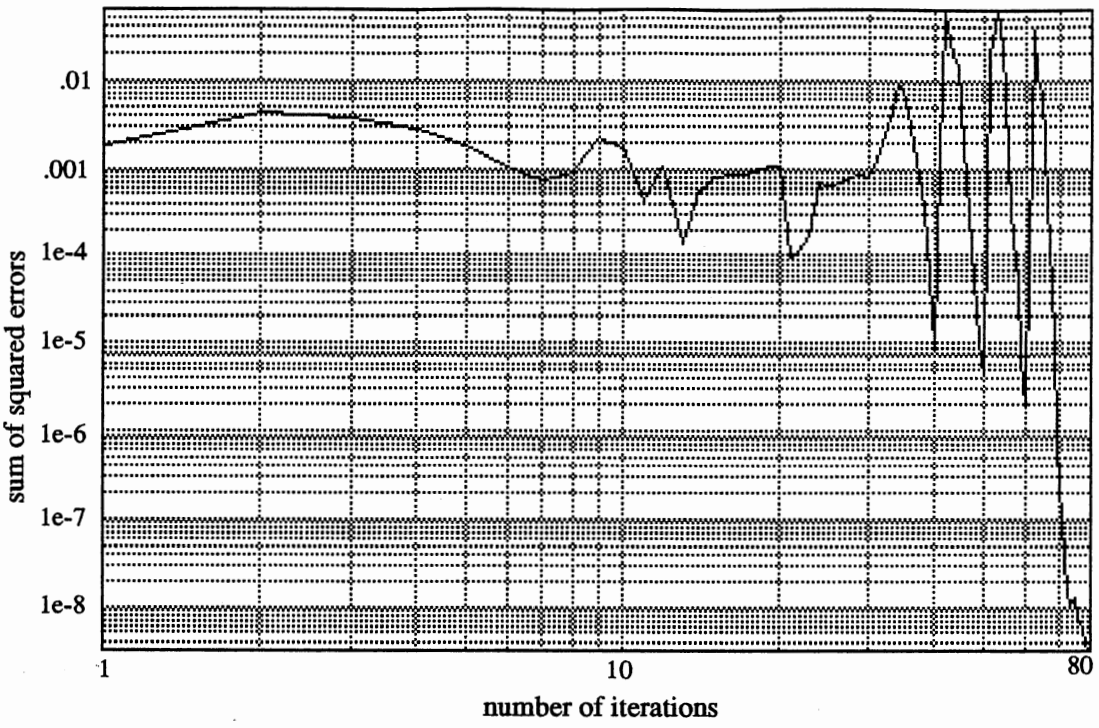


Figure 6.19. Learning Curve for Neural Network N_c (Δt=0.1)

This controller was then used to control the plant. Figure 6.8 shows the operational mode of the system. The response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 0.5 radians when a constant reference input of $\pi/2$ is applied is illustrated in Figure 6.20. The controlled input is u and dx_2/dt is the acceleration of the plant.

Next, a sampling interval of 0.01 sec is used. A neural network N_p is trained to imitate the plant. The learning curve for N_p is shown in Figure 6.21. This neural network is then used to define a controller for the plant. The learning curve for N_c is shown in Figure 6.22.

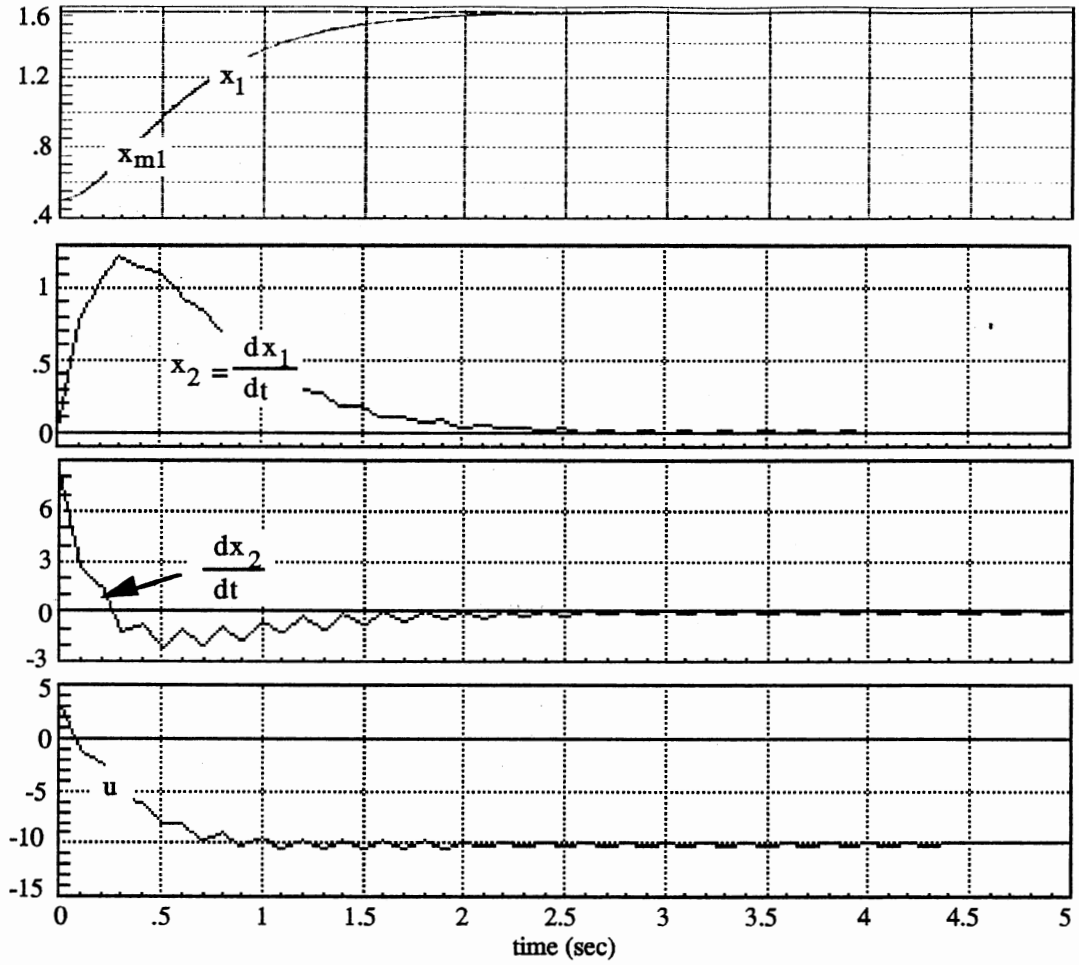


Figure 6.20. Response for Initial Position 0.5 (Δt=0.1)

This controller was then used to control the plant. Figure 6.8 shows the operational mode of the system. The response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 0.5 radians when a constant reference input of $\pi/2$ is applied is illustrated in Figure 6.23. The controlled input is u and dx_2/dt is the acceleration of the plant.

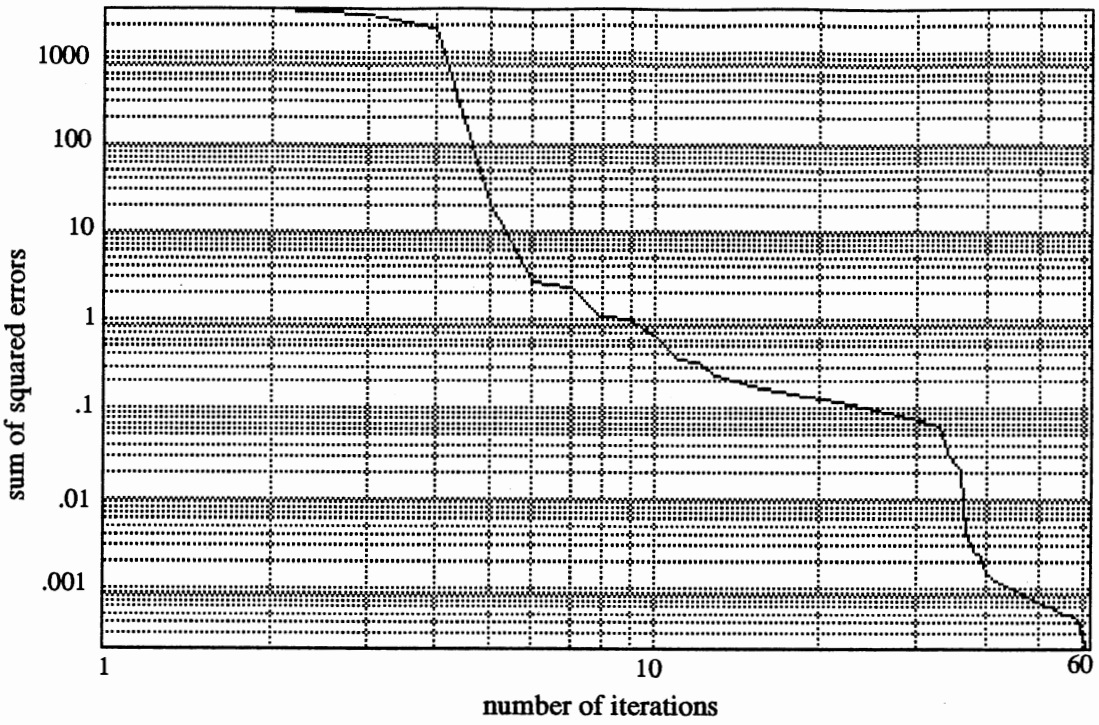


Figure 6.21. Learning Curve for Neural Network N_p (Δt=0.01)

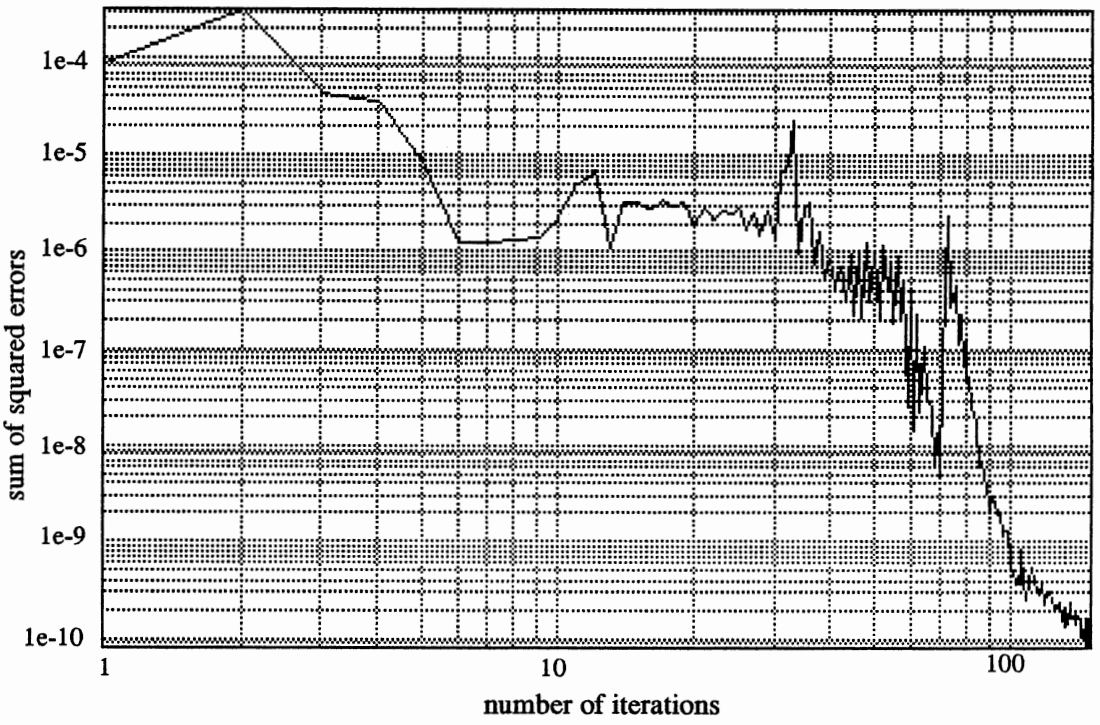


Figure 6.22. Learning Curve for Neural Network N_c (Δt=0.01)

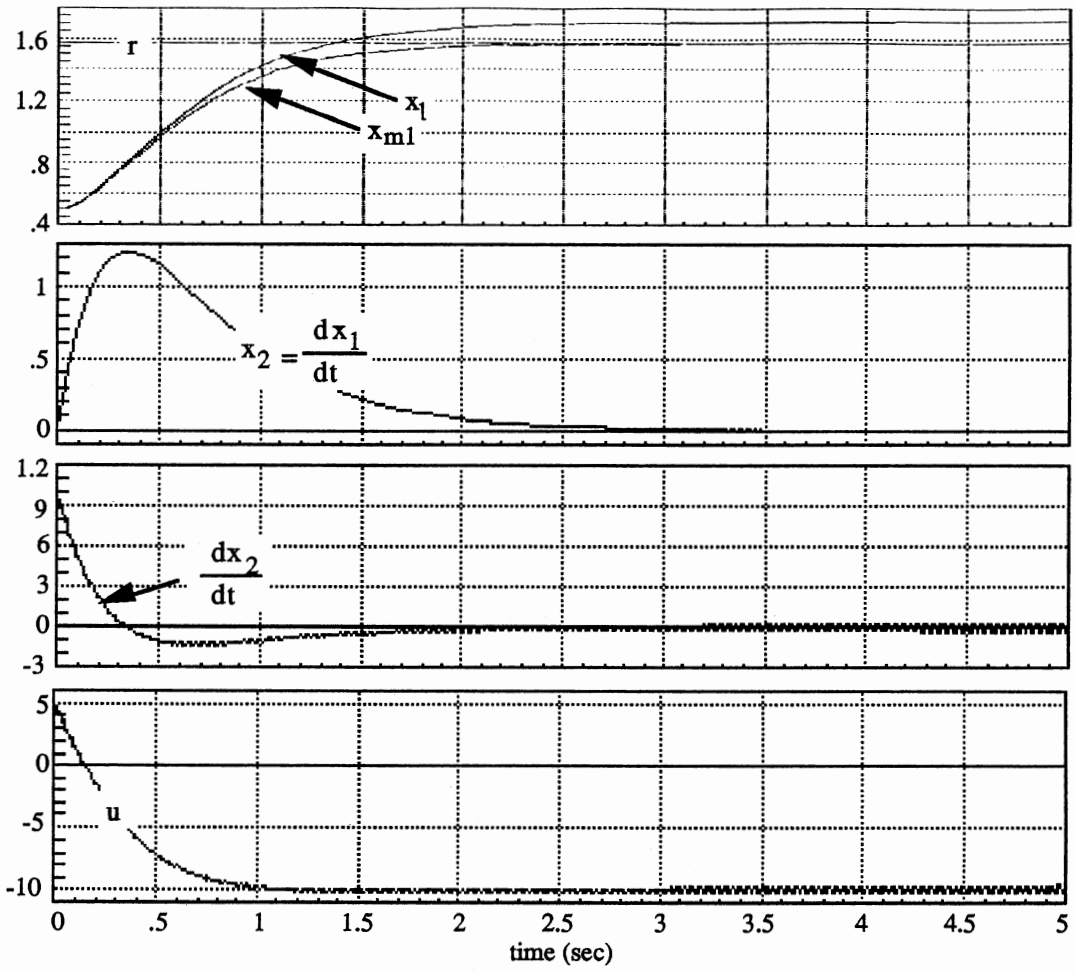
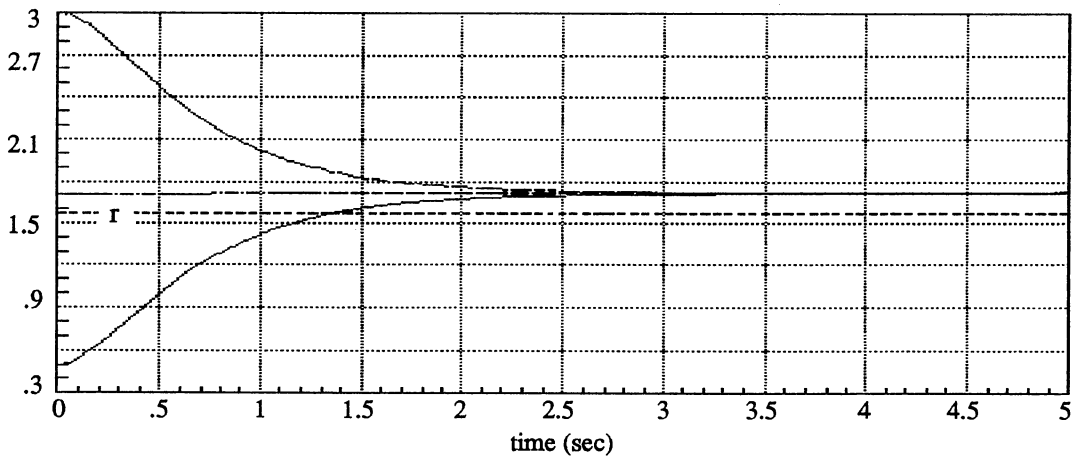


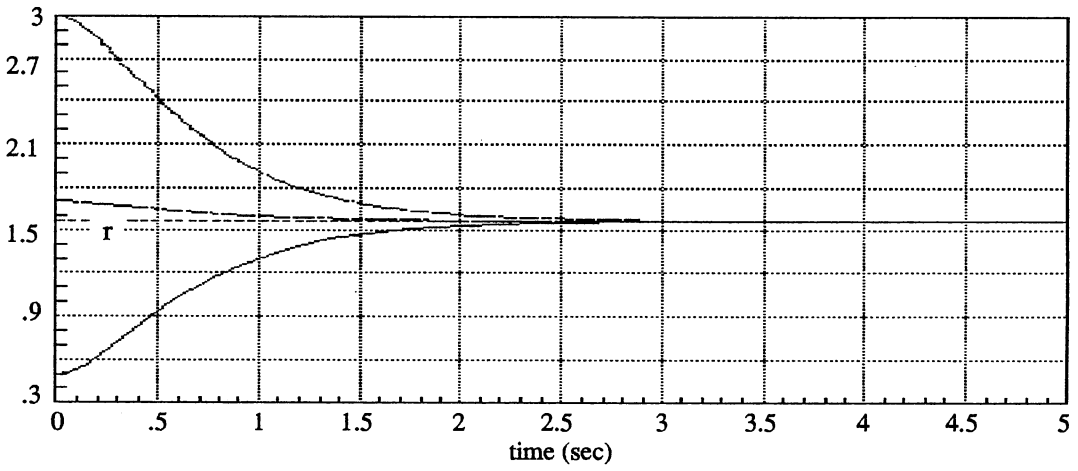
Figure 6.23. Response for Initial Position 0.5 (Δt=0.01)

The controller was able to stabilize the plant, even for a large initial position, but the steady state position is offset from the desired position of $\pi/2$. Figure 6.24 a) shows the response of the system (position only) for three different initial positions (0.5, 1.7 and 3.0). All responses settle to the same steady state. This fact suggests that the bias of the output neuron has not yet been fully learned. After adjusting this bias the response of the system for the three initial positions above is illustrated in Figure 6.24 b). All

positions settle to the desired value, r, of $\pi/2$. This shows the robustness of the controller to initial positions.



a) Before Adjusting the Output Neuron Bias



b) After Adjusting the Output Neuron Bias

Figure 6.24. Response for Initial Position 0.5, 1.7 and 3.0 Radians

Simple Feedback Controller. For comparison purposes a linear controller was used on the elevation model. The linear controller is

a simple feedback controller where just position (the only measurable state) is used as feedback to control the plant. It is designed such that if it were applied to the linearized elevation model the closed loop system would respond like the reference model given above.

The linearized elevation model about a position of $\pi/2$ and a velocity of zero is

$$dx_1/dt = x_2$$

$$dx_2/dt = 10 - 2*x_2 + u$$

or

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ 0 & -2 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 0 \\ 10 \end{bmatrix} = Ax + Bu + D \qquad \text{(VI.35)}$$

Let u be chosen of the form

$$u = -(k*y + H*r + G) = -(F*x + H*r + G) \qquad \text{(VI.36)}$$

where F = [k 0]. Then, substituting u in equation VI.35 gives

$$dx/dt = Ax - B(Fx + Hr + G) + D$$

$$= (A - BF)x - BHr - BG + D \qquad \text{(VI.37)}$$

If G = 10 then BG = D giving

$$dx/dt = (A-BF)x - BHr$$

Substituting the values for A and B gives

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 \\ -k & -2 \end{bmatrix} x - \begin{bmatrix} 0 \\ H \end{bmatrix} r \qquad \text{(VI.38)}$$

For perfect model following, k (i.e. F) should be obtained such that equation VI.38 would equal equation VI.22 (the reference model). This is not possible since $-2 \neq -6$ (element a_{22}). Therefore, the system cannot respond like the reference model.

The root locus for the above system is shown in Figure 6.25. From this figure, the smallest settling time with no oscillation is achieved when the poles of the closed loop system are located at -1. This requires k = 1. With this value the settling time is 3 sec (=3/1). If H is chosen equal to -1, then the controller is

$$u = -(x_1 - r + 10) \qquad \text{(VI.39)}$$

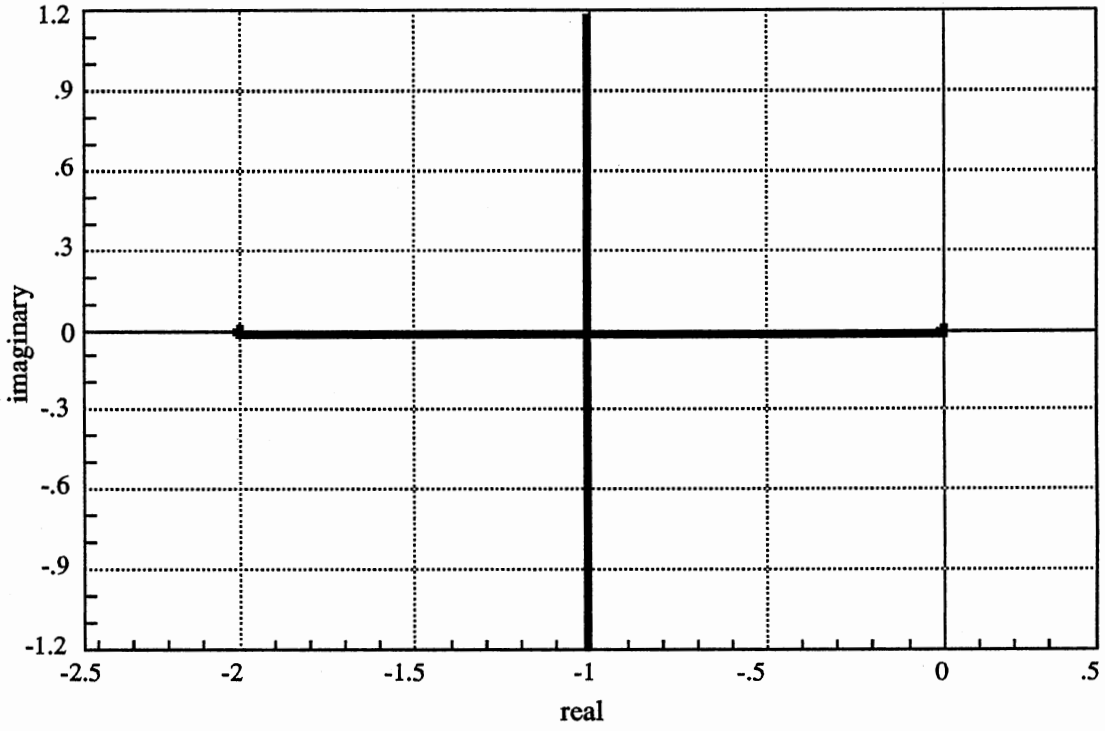
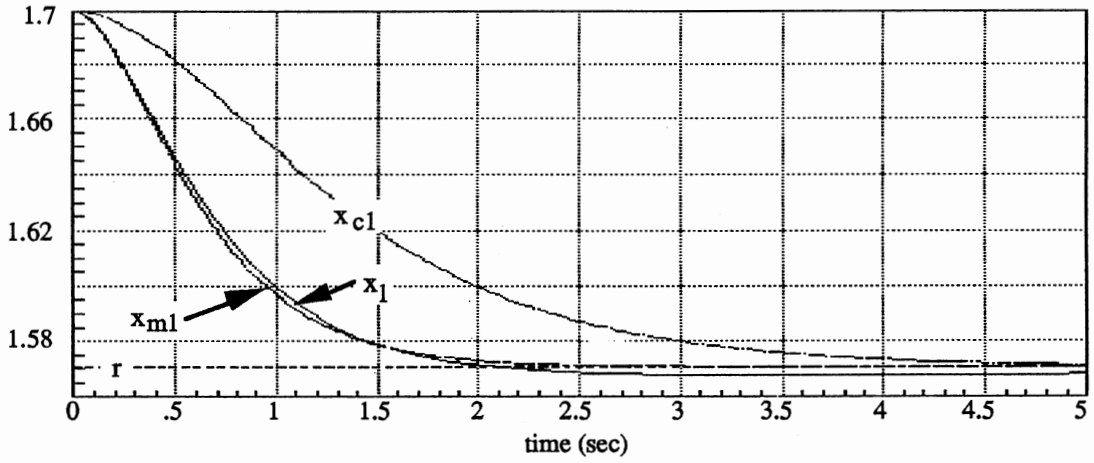


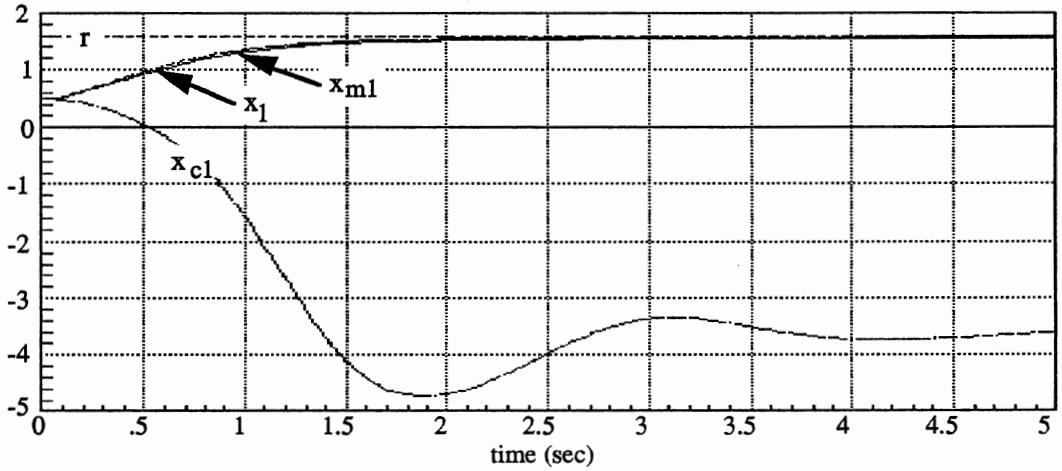
Figure 6.25. Root Locus

Figure 6.26 shows the response (position only) of the neural network controller (x_1), the simple feedback controller (x_{c1}) and the

reference model (x_{m1}) for an initial position of 0.5 and 1.7 radians. As one can see, the neural network controller responds faster than the simple feedback controller for an initial position of 1.7 radians. For an initial position of 0.5 radians the simple feedback controller is unable to stabilize the plant, the plant hits a stop at 0.5 sec.



a) Response for Initial Position 1.7 Radians



b) Response for Initial Position 0.5 Radians

Figure 6.26. Comparison of Controllers

Fourth Order Models

In this section Narendra's general method is applied to design a neural network controller for a fourth order model of the ERG. The fourth order model is obtained when one of the vibrational modes of the ERG is added to the second order systems above.

These vibrational modes, developed because of the flexibility of the gun, are resonances excited by the weapons large unbalance. It affects the pointing accuracy of the weapon which is very strict. This is also found in large space structures. They are distributed parameter systems with a low resonant frequency, a small damping ratio and a high pointing accuracy. Their control has been studied and is still an active area of research [16,17].

Here the resonances are modeled as second order systems with a frequency of 19.8 Hz and a damping ratio of 0.05. It is added to the acceleration of the ERG. These parameters give the following transfer function between the resonant acceleration and input

$$\frac{0.5s^2}{s^2+12.461s+15527} \qquad \text{(VI.40)}$$

which in state variable form is

$$\frac{dv}{dt} = \begin{bmatrix} -12.461 & -121.3047 \\ 128 & 0 \end{bmatrix} v + \begin{bmatrix} 8 \\ 0 \end{bmatrix} u$$

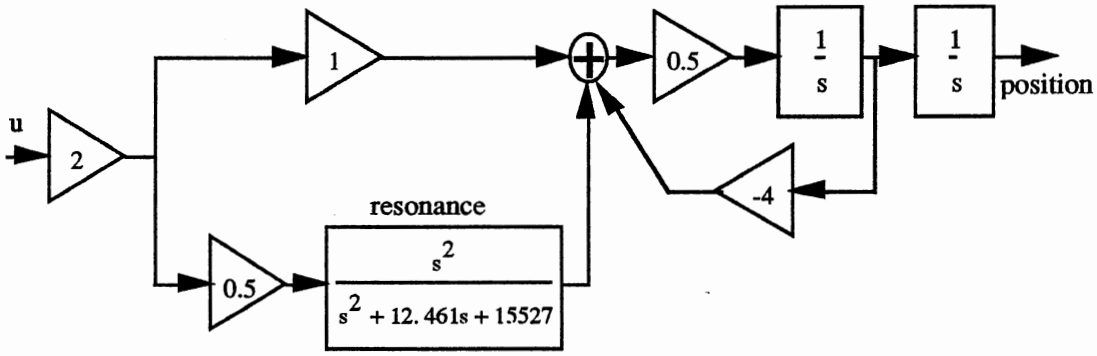
$$w = [-0.7788 \ -7.5815]v + 0.5\ u \qquad \text{(VI.41)}$$

The block diagram in Figure 6.27 illustrates how this resonance is added in the azimuth model and the elevation model given earlier. Here the only measurable state is position. These plants can be described as

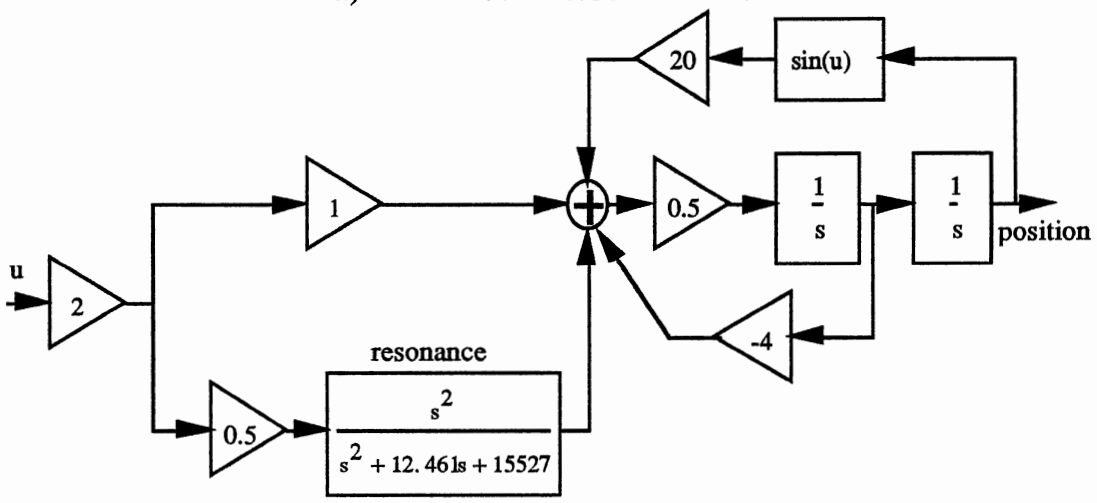
$$y(k+1) = f(y(k),\ y(k-1),\ y(k-2),\ y(k-3),\ u(k),\ u(k-1),\ u(k-2),\ u(k-3)) \quad \text{(VI.42)}$$

and the neural network controller is

$$u(k) = N_c[r(k),\ r(k-1),\ r(k-2),\ r(k-3),\ y(k),\ y(k-1),\ y(k-2),\ y(k-3),\ u(k-1),\ u(k-2),\ u(k-3)] \quad \text{(VI.43)}$$



a) Azimuth Resonant Model



b) Elevation Resonant Model

Figure 6.27. Block Diagram of Resonant System

Next, the neural network controller N_c is designed for the azimuth model. The elevation model is left for future work.

Reference Model

The reference model determines how the output of the plant behaves. The relative order of the reference model should be at least equal to the relative order of the plant (in this case, 4). Therefore, a fourth order reference model was chosen.

The fourth order reference model selected is depicted below. The discrete time transfer function of the second order reference model, equation VI.24, was squared and the constant C obtained such that if a step input were applied the steady state value of y_m would be one. Therefore, the transfer function for the fourth order reference model is

$$\frac{Y_m}{R} = C\left(\frac{(4.411*z+4.3237)*.0001}{z^2-1.9409*z+0.9417}\right)^2$$

$$= \frac{(z^3+2.97z^2+2.9403z+0.970299)*2.0303e-8}{z^4-3.92z^3+5.7624z^2-3.764768z+0.92236816} \qquad \text{(VI.44)}$$

which gives

$$\begin{aligned} y_m(k+1) &= 3.92*y_m(k) - 5.7624*y_m(k-1) + \\ &+ 3.764768*y_m(k-2) - 0.92236816*y_m(k-3) + \\ &+ (r(k) + 2.97*r(k-1) + 2.9403*r(k-2) + \\ &+ 0.970299*r(k-3))*2.0303e-8 \end{aligned} \qquad \text{(VI.45)}$$

Azimuth Model

The azimuth model (θ fixed) is shown in Figure 5.3. Here the second order resonance is added to the second order system giving a fourth order linear system. The equations in state variable form that describe this system are

$$\frac{dx}{dt} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -2 & -0.3894 & -3.7908 \\ 0 & 0 & -12.461 & -121.3047 \\ 0 & 0 & 128 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1.5 \\ 16 \\ 0 \end{bmatrix} u \qquad \text{(VI.46)}$$

where: $x_1 = \phi$

$x_2 = d\phi/dt$

u - input current to the motor that moves the gun in ϕ

The method used for integration is third-order Runge-Kutta (see Appendix) with Δt=0.01 sec, the sampling interval.

As described above, the first stage in designing the controller is to obtain a neural network that imitates the plant. Figure 6.6 shows a sketch of the procedure for plant identification.

The plant can be described as in equation VI.42. A neural network Np with one layer of linear neurons having eight inputs (y(k), y(k-1), y(k-2), y(k-3), u(k), u(k-1), u(k-2), u(k-3)) and one output (y(k+1)) was trained to mimic the plant. The learning curve for N_p is shown in Figure 6.28. This neural network was then used to define a controller for the plant.

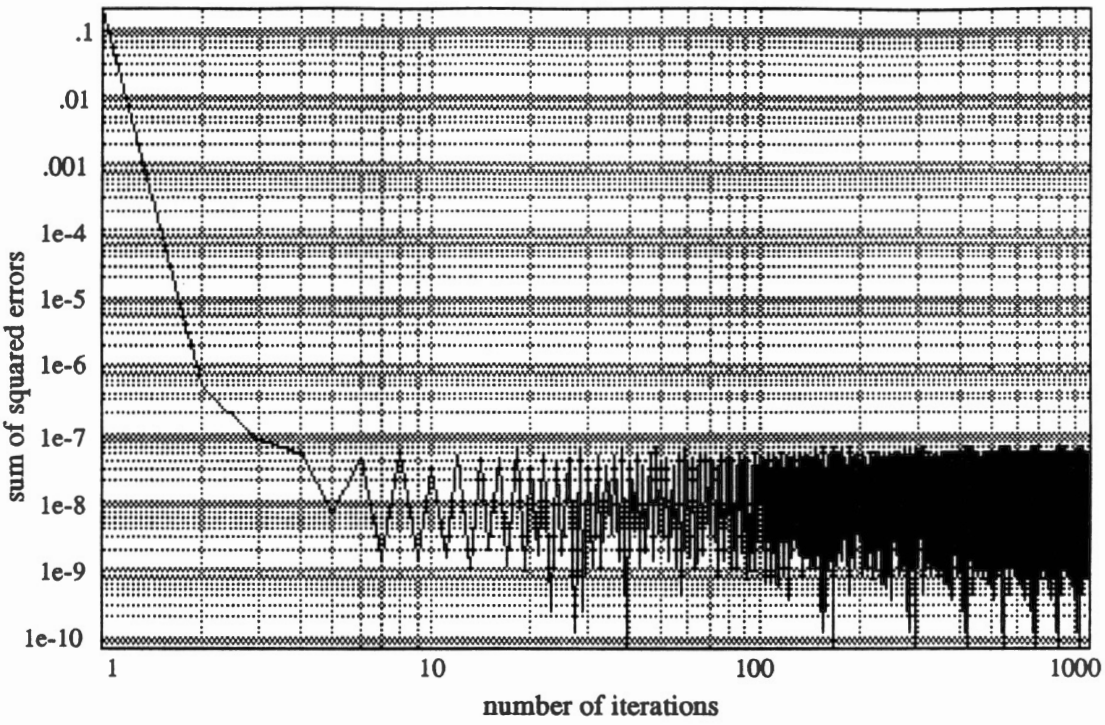


Figure 6.28. Learning Curve for Neural Network N_p

The controller is detailed in equation VI.43. Neural network N_c has one layer of linear neurons with eleven inputs (y(k), y(k-1), y(k-2), y(k-3), u(k-1), u(k-2), u(k-3), r(k), r(k-1), r(k-2), r(k-3)) and one output (u(k)). It was trained to control the plant. The learning curve for N_c is shown in Figure 6.29.

This controller was then used to control the plant. Figure 6.8 shows the operational mode of the system. The response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 3.0 radians when a constant reference input of zero is applied is illustrated in Figure 6.30. The controlled input is u and dx_2/dt is the acceleration of the plant. As one can see, the

controller was able to stabilize the plant and also damp out the resonance improving pointing accuracy.

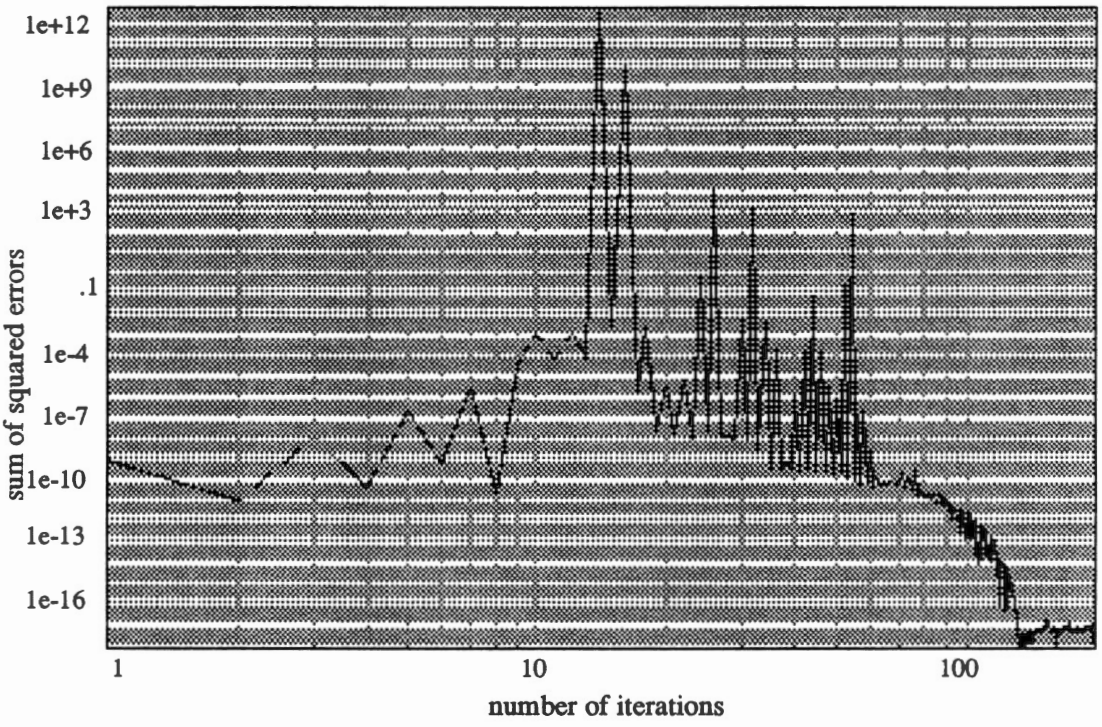


Figure 6.29. Learning Curve for Neural Network N_c

Simple Feedback Controller. For comparison purposes a linear controller was used on the azimuth model. The linear controller is a simple feedback controller where just position (the only measurable state) is used as feedback to control the plant. It is designed such that the fastest settling time without oscillation is obtained.

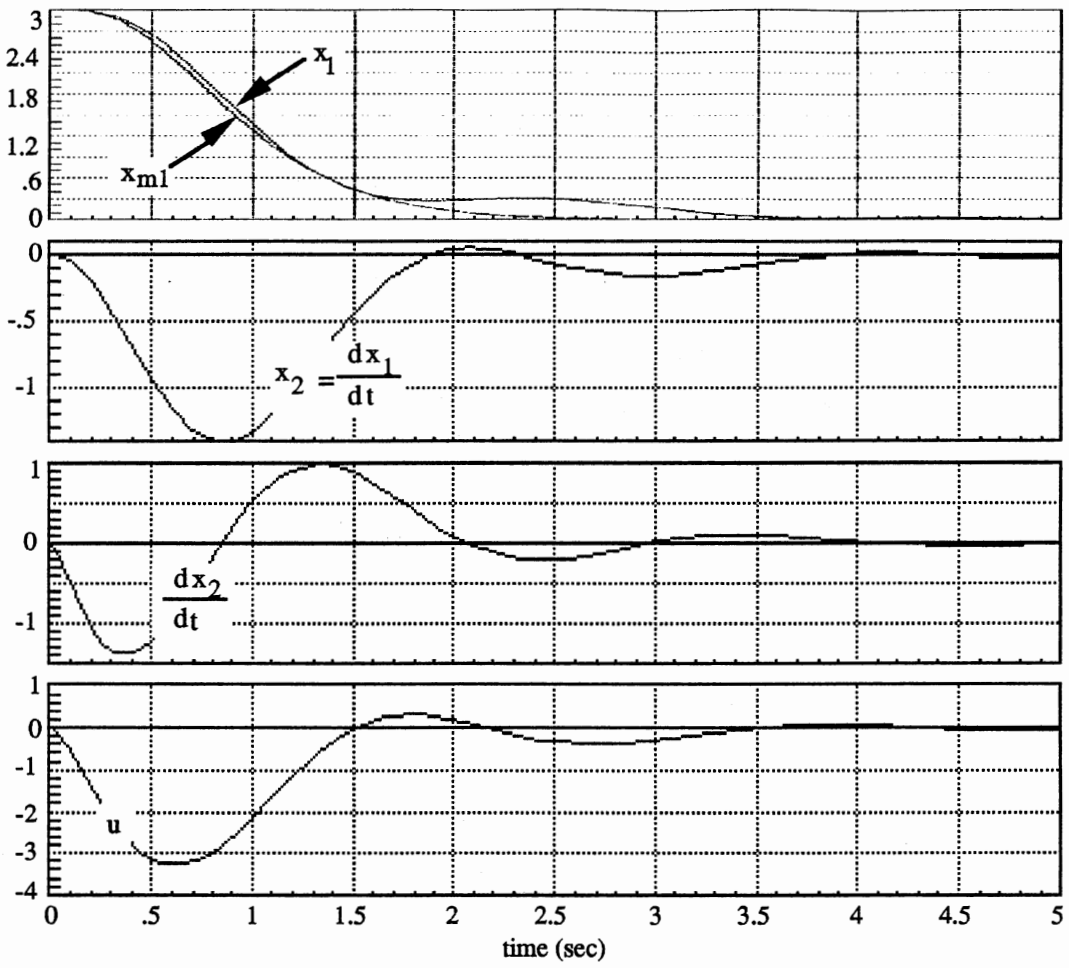


Figure 6.30. Response for Initial Position 3.0

Let u be chosen of the form

$$u = -(k*y + H*r) \qquad \text{(VI.47)}$$

The root locus for the azimuth model is shown in Figure 6.31. From this figure, the smallest settling time with no oscillation is achieved when the poles of the closed loop system are located at -1. This requires k = 1. If H is chosen equal to -1, then the controller is

$$u = -(x_1 - r) \qquad \text{(VI.48)}$$

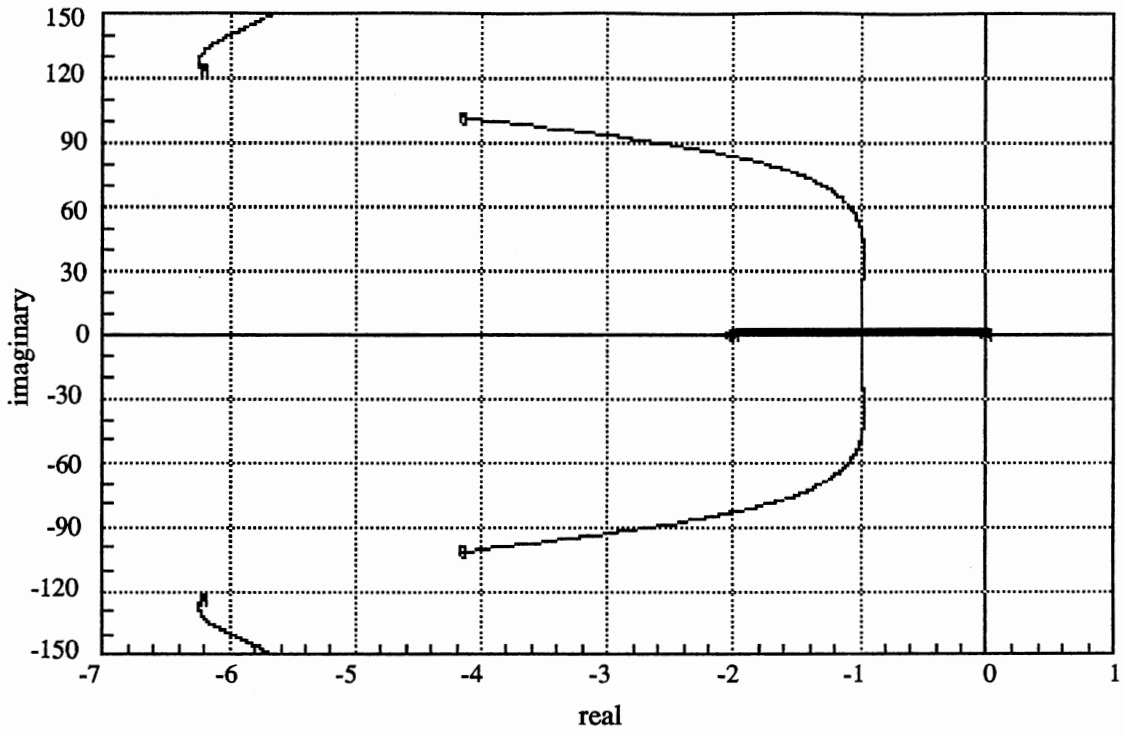


Figure 6.31. Root Locus

Figure 6.32 shows the response of the system (x_1 and x_2) and the reference model (x_{m1}) to an initial position of 3.0 radians when a constant reference input of zero is applied. The controlled input is u and dx_2/dt is the acceleration of the plant. As one can see, the controller was able to stabilize the plant but was mot capable of damping out the resonance in the acceleration.

In the next chapter, a summary and topics for future work are presented. Future work includes applying the general method to a controller for the elevation model of the ERG with resonance.

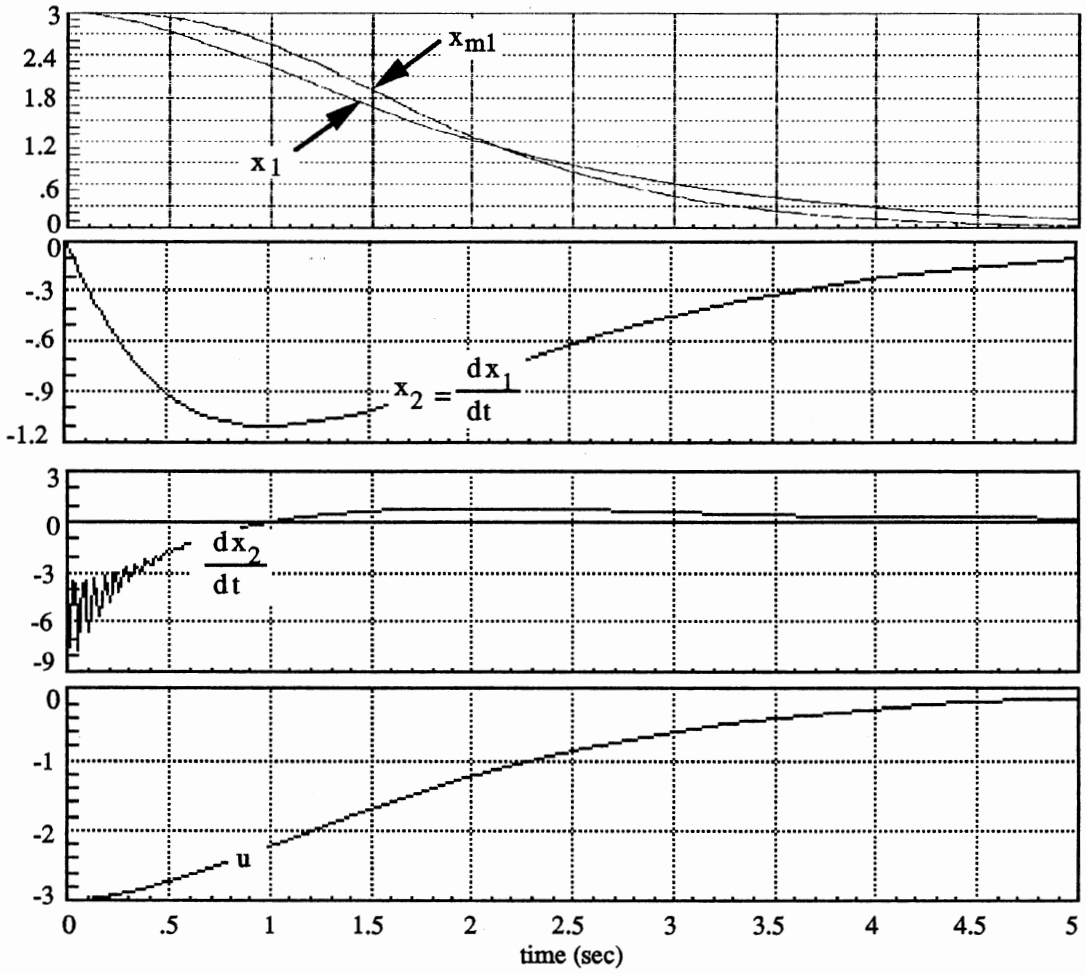


Figure 6.32. Response for Initial Position 3.0 (Simple Controller)

CHAPTER VII

SUMMARY AND FUTURE WORK

This research investigated the use of neural networks as controllers for dynamical systems, with particular emphasis on the stabilization of the Extended Range Gun (ERG). Two design techniques using neural network controllers, developed by Widrow and Narendra, have been studied. The backpropagation algorithm, which is used in both techniques, was explained in detail in Chapter II.

Chapter III described modifications to the backpropagation algorithm to reduce training time. Some reduction in training time was obtained, but it was still high. Therefore, another algorithm, the conjugate gradient with line search, was suggested. This algorithm decreased the number of iterations a significant amount.

The use of neural networks in control systems was described in Chapter IV. Here, the two techniques used to design neural network controllers were explained. Widrow developed the first technique while the second one was established by Narendra. In Chapter V Widrow's technique was used to control a simple model of the ERG. Narendra's feedback linearization method used on the simple model was presented in Chapter VI. Results showed that both techniques were able to provide good control of the weapon.

Chapter VI also included defining a controller for the simple model of the ERG using the general method developed by Narendra. With this training method, the conjugate gradient algorithm was not able, in many cases, to determine a controller for the gun. A variation of the backpropagation algorithm, the Levenberg-Marquardt algorithm, was capable of finding a controller for the gun.

In this research only the linear azimuth model with one resonant mode was included. More complex models were left as future work. These models would include the addition of more resonant modes to the simple gun systems used in this research. Another aspect to be explored in future research would be to find a controller for the nonlinear elevation model with resonances. Another area of future work is on-line learning. When the modes cannot be adequately modeled, the controller could adjust to the unmodeled dynamics after installation. Narendra's method is promising in this area.

REFERENCES

[1] R.P. Lippmann, "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, vol. 4, no. 2, p 4-22, April 1987.

[2] R. Hecht-Nielsen, "Theory of the Backpropagation Neural Network," presented at the International Neural Networks Society Annual Meeting, Sept 1988.

[3] R. Hecht-Nielsen, *Neurocomputing*. Addison-Wesley (1990).

[4] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*. MIT Press (1986).

[5] D.M. Himmelblau, *Applied Nonlinear Programming*. McGraw-Hill (1972).

[6] T.P Vogl, et. al., "Accelerating the Convergence of the Backpropagation Method," Biological Cybernetics, vol. 59, p 257-263, 1988.

[7] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Network by Choosing Initial Values of the Adaptive Weights," Proceedings of the IEEE International Joint Conference on Neural Networks, vol. III, p.21-26, July 1990.

[8] L.E. Scales, *Introduction to Non-Linear Optimization*. Springer-Verlag (1985).

[9] R. Fletcher and C.M. Reeves, "Function Minimization by Conjugate Gradients," Computer Journal, vol. 7, no. 2, p 149-153, 1964.

[10] W.L. Brogan, *Modern Control Theory*. Prentice-Hall (1985).

[11] A. Guez and J. Selinsky, "A Trainable Neuromorphic Controller," Journal of Robotic Systems, vol. 5, no. 4, p 363-388, 1988.

[12] D. Nguyen and B. Widrow," The Truck Backer-Upper: An Example of Self-Learning in Neural Networks," Proceedings of the IEEE International Joint Conference on Neural Networks, vol II, p.357-363, 1989.

[13] D. Nguyen and B. Widrow," Neural Networks for Self-Learning Control Systems," IEEE Control Systems Magazine, vol. 10, no. 3, p 18-23, April 1990.

[14] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," IEEE Transactions on Neural Networks, vol. 1, no. 1, p 4-27, March 1990.

[15] K.S. Narendra, E.G. Kraft and L.H. Ungar, *Neural Networks in Control Systems*, workshop at the American Control Conference, Boston, MA, May 1991.

[16] M.J. Balas, "Trends in Large Space Structure Control Theory: Fondest Hopes, Wildest Dreams," IEEE Transactions on Automatic Control, vol. 27, no. 3, p 522-535, June 1982.

[17] E.A. Czajkowski, A. Preumont and R.T. Haftka, "Spillover Stabilization of Large Space Structures," Journal of Guidance, Control, and Dynamics, vol.13, no. 6, p 1000-1007, Nov-Dec 1990.

[18] B. Carnahan, H.A. Luther and J.O. Wilkes, *Applied Numerical Methods*. John Wiley & Sons, Inc (1969).

APPENDIX

THIRD-ORDER RUNGE-KUTTA INTEGRATION METHOD

The method of integration used to obtain the solution of the differential equation describing the plant was third-order Runge-Kutta [18]. This algorithm gives approximations as accurate as higher-order Taylor formulas but only include first-order derivatives.

The third-order approximation is equivalent in precision to Taylor's expansion that include terms up to $(\Delta t)^3$. It requires the evaluation of the function at three points in the interval $[t, t+\Delta t]$. If

$$\dot{x} = f(t,x)$$

then the approximation is of the form

$$x_{i+1} = x_i + \Delta t * \phi(t_i, x_i, \Delta t) \qquad (A.1)$$

where ϕ is the increment function by Henrici, an approximation to $f(t,x)$.

Let ϕ be a weighted sum of k_1, k_2 and k_3, derivative evaluations on the interval $[t,t+\Delta t]$

$$\phi = a*k_1 + b*k_2 + c*k_3 \qquad (A.2)$$

Substituting in equation A.1 gives

$$x_{i+1} = x_i + \Delta t*(a*k_1 + b*k_2 + c*k_3) \qquad (A.3)$$

Let

$$k_1 = f(t, x_i)$$

$$k_2 = f(t + p*\Delta t, x_i + p*\Delta t*k_1)$$

$$k_3 = f(t + r*\Delta t, x_i + s*\Delta t*k_1 + (r-s)*\Delta t*k_2) \qquad (A.4)$$

where p, r and s are constants.

These constants are obtained by expanding k_2, k_3 and f( ) about (t,x_i) in a Taylor's series, dropping the terms having the exponent of Δt greater than 3 and equating the expressions. The following equations are then obtained

$$a + b + c = 1$$

$$b*p + c*r = 1/2$$

$$b*p^2 + c*r^2 = 1/3$$

$$c*p*s = 1/6 \qquad (A.5)$$

There are more unknowns (6) (a, b, c, p, r, s) than equations (4), therefore two of the constants are chosen arbitrarily. If the constant values are

$$a = 1/6,\ b = 1/6,\ c = 4/6,\ p = 1,\ r = 1/2,\ s = 1/4$$

then the third-order Runge-Kutta is given by

$$x_{i+1} = x_i + \Delta t*(k_1 + k_2 + 4*k_3)/6 \qquad (A.6)$$

where

$$k_1 = f(t, x_i)$$

$$k_2 = f(t + \Delta t, x_i + \Delta t*k_1)$$

$$k_3 = f(t + \Delta t/2,\ x_i + (1/4)*\Delta t*k_1 + (1/2-1/4)*\Delta t*k_2)$$

$$= f(t + \Delta t/2,\ x_i + (1/4)*\Delta t*(k_1 + k_2))$$

The above equation is the formula used for integration of the differential equation describing the plant specified in Chapter VI.

VITA

Gisele Guimarães

Candidate for the Degree of

Doctor of Philosophy

Thesis: NEURAL NETWORKS FOR CONTROL

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Goiânia, Goiás, Brazil, January 31, 1961, the daughter of Jerson D. Guimarães and Balbina A.S. Guimarães.

Education: Received the Bachelor of Science degree in Electrical Engineering from Universidade Federal de Goiás, Goiânia, Goiás, Brazil, in October 1984; received the Master of Science degree in Electrical Engineering from Oklahoma State University in December 1988; completed the requirements for the Doctor of Philosophy degree at Oklahoma State University in July1992.

Professional Experience: Graduate Research Assistant, Department of Electrical Engineering, Oklahoma State University, from January 1988; Teaching Assistant, Department of Electrical Engineering, Oklahoma State University, January 1989 to May 1989; Software implementation and student assistant for a graduate course at Universidade Católica de Goiás, Goiânia, Goiás, Brazil, March 1985 to December 1985.

Membership in Professional Societies: Institute of Electrical and Electronics Engineers in Control Systems, Neural Networks, Computer Societies.

Membership in Honorary Societies: Tau Beta Pi and Eta Kappa Nu.